MASTER EN SCIENCES INFORMATIQUES

Réalisation d'un progiciel graphique

Mataigne, Ph.; Perozzo, Y.

Award date:
1982

Awarding institution:
Universite de Namur

FACULTES UNIVERSITAIRES NOTRE-DAME DE LA PAIX (NAMUR)

INSTITUT D'INFORMATIQUE

REALISATION

D'UN PROGICIEL

GRAPHIQUE

Mémoire présenté par
Ph. Mataigne
Y. Perozzo
en vue de l'obtention
du titre de
Licencié et Maitre en Informatique

Année académique 1981-1982

Nous remercions vivement Monsieur Jean Fichefet d'avoir accepté la direction de ce mémoire.

Nous remercions également Patrick Lambion, dont les remarques et les conseils nous ont permis de mener à bien la réalisation du projet.

Nous remercions enfin tous les membres de l'Institut et du Centre de Calcul, qui de près ou de loin, ont contribué à l'élaboration de ce travail.

TABLE DES MATIERES
++++++++++++++++++++

C H A P I T R E 1

INTRODUCTION GENERALE.

1.1 Identification du problème.

Dans le cadre de ses activités, l'UER Mathématiques appliquées de l'Institut d'Informatique traite fréquemment des applications scientifiques faisant intervenir de nombreuses données et nécessitant d'importants moyens de calcul.

Si les programmes utilisés pour traiter ces applications sont actuellement bien au point, ils laissent toutefois apparaître des lacunes importantes dans la présentation des résultats. Cette présentation est souvent rudimentaire et peu explicative.

Ainsi, les utilisateurs de tels programmes ont-ils fréquemment recours à des graphiques tracés à la main pour faciliter l'interpretation de ces résultats. C'est pourquoi, pour tenter d'améliorer cette situation, nous avons envisagé la création d'un progiciel graphique.

A ce stade, il faut apporter quelques précisions quant au contenu de ce progiciel.

Premièrement, étant donné le temps qui nous était imparti, nous avons dû limiter notre travail aux sujets suivants :

- représentation et manipulation de graphes,
- tracé d'histogrammes et de fonctions d'une variable,
- représentation de nuages de points dans le cadre de l'analyse en composantes principales,
- représentation de classifications hiérarchiques.

Deuxièmement, le progiciel se présente sous la forme d'une librairie de sous-programmes graphiques et non pas d'un ensemble de programmes traitant des applications complètes. L'utilisateur doit donc écrire lui-même un programme d'application dans lequel il prend en charge l'obtention et la mise en forme des données avant d'appeler les routines graphiques du progiciel.

1.2 Etapes du travail.

A la suite de cette introduction générale, nous aborderons les chapitres suivants :

- Le chapitre 2 est consacré aux outils qui constituent la base de ce mémoire. Nous verrons d'une part le matériel auquel nous avons eu accès (table traçante et terminal graphique), et d'autre part un logiciel graphique appelé DI-3000, dont le Centre de calcul a fait l'acquisition.
- Le chapitre 3 est consacré dans sa première partie aux problèmes consécutifs au choix du langage de programmation. La seconde partie reprend les solutions qui ont été apportées aux problèmes faisant l'objet de ce mémoire.
- Enfin, le chapitre 4 est consacré à l'évaluation de ces solutions.

Le lecteur trouvera également, à la suite du chapitre 4, la bibliographie des ouvrages que nous avons consultés.

Les annexes ont été, quant a elles, séparées en deux parties.
L'annexe A constitue le manuel d'utilisation des procédures de la librairie.
L'annexe B reprend les procédures elles-mêmes.

C H A P I T R E 2

OUTILS GRAPHIQUES DE BASE.

2.1. Matériel.

Pour la réalisation de ce mémoire, nous disposions d'un environnement graphique constitué de deux appareils, un terminal graphique Digital Equipment et une table traçante Philips, dont nous allons décrire succintement les caractéristiques.

* Le terminal

Il s'agit d'un terminal intelligent de type GIGI (General Imaging Generator and Interpreter), qui peut travailler en huit couleurs.

Relié à un moniteur standard noir et blanc ou couleur, et à un ordinateur hôte, le GIGI peut travailler en deux modes distincts :

a) soit en tant que terminal standard de l'hôte (on dit qu'il est en mode texte), avec un écran constitué de 24 lignes de 80 caractères, (fig. 1)

b) soit en tant que terminal graphique intelligent (il est alors en mode graphique).

Dans ce dernier cas, c'est un microprocesseur intégré dans le terminal qui se charge de l'interprétation des commandes graphiques en provenance de l'hôte et de l'affichage du dessin sur le moniteur.

Dans le mode graphique, le GIGI a une résolution de 758 points horizontaux sur 240 points verticaux.

* La table traçante (modèle PM 8151).

Elle permet l'utilisation de huit marqueurs de couleur différente, mais actuellement seuls quatre d'entre eux sont disponibles.

Sa résolution est de 0.1 millimètres.

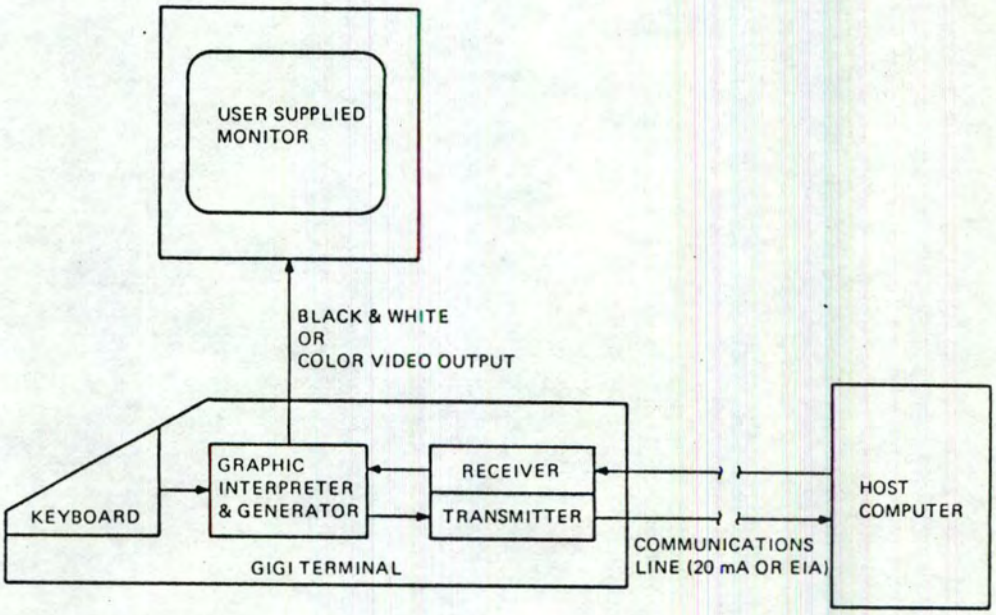


FIGURE 1
Terminal en mode graphique

Elle accepte des feuilles de taille A3 (297 X 420 millimètres), et le dessin occupe au maximum un espace de 280 X 338 millimètres.

2.2. Logiciel.

L'exposé est subdivisé en trois parties :

- La présentation générale du logiciel DI-3000,
- La description d'un outil particulier, le métafichier DI-3000,
- Un glossaire des termes techniques couramment utilisés dans la description des logiciels graphiques.

Par souci de concision, seules sont abordées les notions qui peuvent aider à une meilleure compréhension de la suite de notre travail. Le lecteur désireux d'approfondir la connaissance de ce logiciel se référera utilement à [DI].

2.2.1. DI-3000 : Présentation générale.

DI-3000 est un ensemble d'outils logiciels graphiques écrit dans le langage Fortan Ansi 1966. Il est utilisé sous la forme d'une librairie de sous-programmes pouvant être appelés à partir d'un programme écrit en Fortan.

A. Philosophie de conception

La conception de DI-3000 est basée sur l'idée qu'un programme générant des objets graphiques doit être indépendant de l'environnement dans lequel ce programme s'exécute (en anglais, device independent). Par environnement, nous entendons l'ensemble des appareils (écran vidéo, table traçante, tablette de digitalisation, ...) qui constituent habituellement une installation graphique.

NOTE : Les [ ] renvoient à la bibliographie.

Cette conception signifie, en particulier, qu'un seul programme devrait pouvoir sortir une même image sur plusieurs appareils graphiques aux caractéristiques différentes.

Pour illustrer cette notion d'indépendance, nous nous référerons aux figures 2 , 3 , 4 . La première figure montre deux logiciels graphiques développés pour des appareils différents, chacun présentant un interface distinct aux programmes d'application qui sont donc nécessairement différents. De tels logiciels obligent le programmeur a tenir compte des différents aspects offerts par les appareils graphiques.

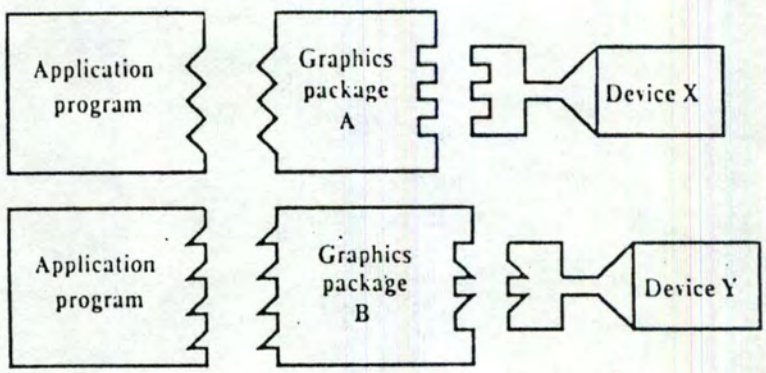


FIGURE 2

La seconde figure montre une tentative pour réaliser un interface uniforme pour le programmeur. Le même programme d'application peut maintenant être exécuté dans deux environnements différents. Nous avons obtenu l'indépendance vis-à-vis du programmeur, mais les deux logiciels graphiques diffèrent encore de façon interne.

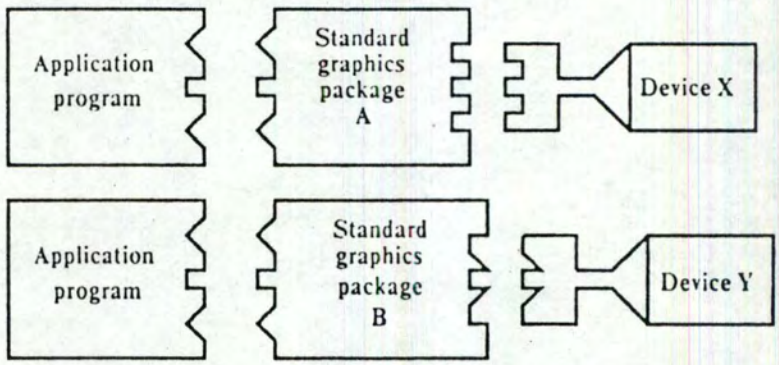


FIGURE 3

La troisième figure fait apparaître la notion de contrôleur d'appareil (device driver). Ce contrôleur présente un interface uniforme au logiciel graphique et permet de considérer les appareils d'entrée ou de sortie graphique comme des appareils virtuels, en quelque sorte idéalisés (virtual graphics device). Il y a ainsi indépendance vis-à-vis du programmeur et indépendance vis-à-vis du logiciel.

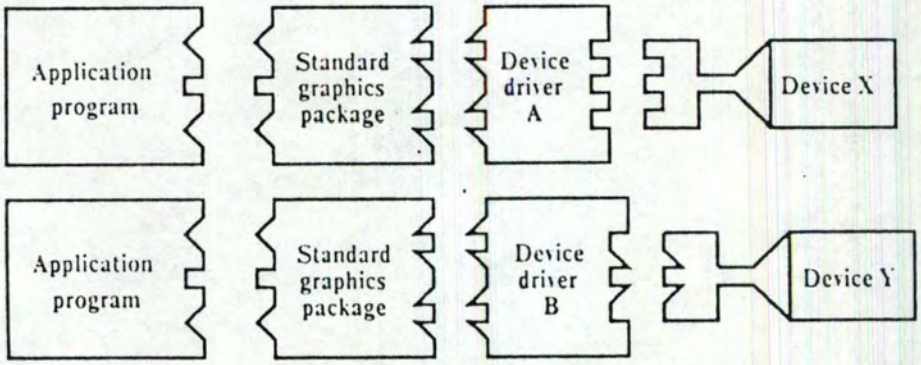


FIGURE 4

B. Le réseau DI-3000.

DI-3000 est conçu comme un réseau logiciel modulaire (fig. 5). Le programme d'application ne fait appel qu'aux sous-programmes graphiques afin de générer des commandes indépendantes des appareils spécifiques (et donc destinées à des appareils virtuels qui les supportent).

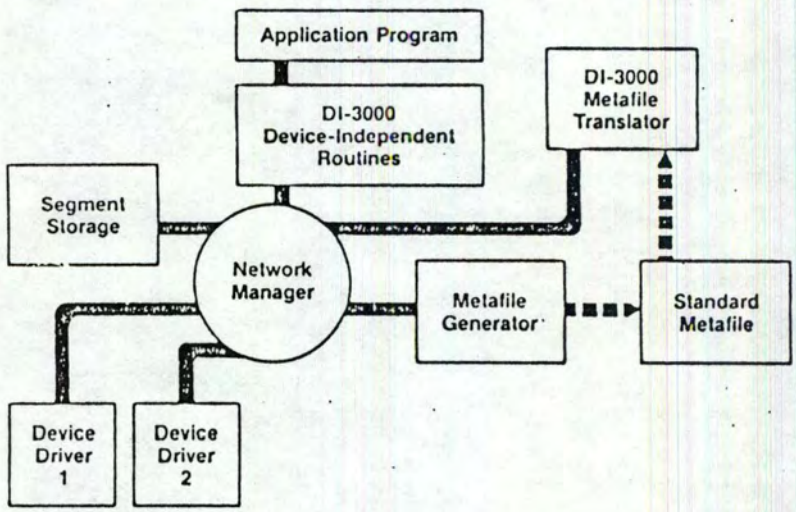


FIGURE 5
Le réseau DI-3000.

Au sein d'un environnement graphique, chaque appareil (spécifique ou virtuel) est supporté par un contrôleur. Ce dernier est une librairie de sous-programmes qui interprètent les commandes graphiques et les convertissent en des instructions permettant de piloter un appareil spécifique.

En tant qu'appareil idéalisé, un appareil virtuel offre souvent des possibilités qui ne sont pas directement celles d'un appareil physique. Le contrôleur peut alors ignorer ou simuler ces possibilités par logiciel.

Enfin, étant donné qu'un même programme d'application peut accéder sélectivement à un ou plusieurs appreils graphiques, un gérant du réseau (network manager) garantit que les messages provenant du programme sont correctement acheminés vers le contrôleur de l'appareil correspondant.

Les autres composantes de ce réseau, à savoir l'espace des segments (segment storage) et tout ce qui concerne le métafichier (metafile) seront présentés par la suite.

C. Un modèle conceptuel pour DI-3000.

L'indépendance de DI-3000 vis-à-vis d'un environnement graphique repose sur les concepts suivants :

1. Le programme d'application fait référence à un ou plusieurs appareils virtuels. Au moment de l'exécution, le programmeur nomme un ou plusieurs appareils physiques qui remplaceront ces appareils virtuels.

2. Le programme d'application décrit des objets dans un espace à deux ou trois dimensions à l'aide de primitives graphiques, par exemple des lignes, des chaînes de caractères ou des symboles spéciaux (markers).

3. Ces primitives sont définies dans un système de coordonnées réelles à trois dimensions (world coordinate system), dont les unités et les dimensions sont établies par le programme d'application.

4. Les primitives peuvent être regroupées dans des segments, une image étant composée d'un ou plusieurs segments mutuellement exclusifs.

5. Le programme d'application peut assigner un nom à un segment. Un segment nommé est appelé un segment retenu (retained segment) et est mémorisé dans l'espace des segments. Nous verrons plus loin l'utilité de tels segments.

6. Un appareil graphique virtuel possède une région adressable dont les unités en X et Y varient entre -1 et 1. Cette région est appelée le système de coordonnées virtuelles ou normalisées (normalized device coordinate system).

7. Le programme d'application définit la correspondance entre une région rectangulaire, la fenêtre, dans le système de coordonnées réelles et une région rectangulaire, le champ de vision (viewport), dans le système de coordonnées virtuelles. Cette correspondance est appelée la conversion d'image (viewing transformation).

   La conversion d'image est également utilisée pour définir la position et l'orientation d'une camera virtuelle dans un système de coordonnées à trois dimensions.

D. Résumé fonctionnel.

Dans ce paragraphe, nous passons en revue les possibilités fontionnelles de DI-3000. L'énumération des fonctions qui réalisent ces possibilités étant trop longue, nous renvoyons le lecteur a [DI], appendice A.

1. Primitives de tracé et positionnement.

Ces primitives définissent les déplacements "plume haute" (moves), les déplacements "plume basse" (draws) pour le tracé de lignes, de polylignes, de polygones et de symboles spéciaux (markers).

La position courante est un point de référence dans le système de coordonnées réelles. Le déplacement "plume haute" modifie la position courante. Le déplacement "plume basse" trace une ligne depuis la position courante jusqu'à un autre point dont on précise les coordonnées réelles. Ce point deviendra la nouvelle position courante.

La primitive polyligne permet de joindre par un ou plusieurs segments de droite un ensemble de points dont on fournit les coordonnées réelles.

La primitive polygone permet de définir une région fermée (telle qu'un cercle, un secteur ou un rectangle) à l'aide de trois points au moins.

DI-3000 offre également la possibilité de tracer certains symboles spéciaux standards (astérisque, diamant) en la position courante.

Enfin, chaque positionnement ou primitive de tracé peut s'effectuer en coordonnées réelles, relatives ou absolues, dans un espace à deux ou trois dimensions.

2. Attributs des primitives de tracé et de positionnement.

Ces attributs sont utilisés pour définir les caractéristiques générales des primitives. Il s'agit principalement de :

- la couleur du trait,
- l'intensité ou brillance relative du trait,
- le style de ligne (solide, pointillé, ...),
- la largeur du trait,
- le style du bord d'un polygone,
- le style intérieur d'un polygone (vide, hachuré, ...),
- la couleur ou l'intensité intérieure d'un polygone,
- le type de symbole spécial (astérisque, diamant, ...),

Chaque attribut peut avoir des valeurs par défaut.

3. Primitives de texte.

Afin de permettre au programmeur d'application de nommer un dessin ou de commenter un graphique, DI-3000 fournit des primitives de texte pour le tracé de chaînes de caractères.

Plusieurs qualités de texte sont disponibles et peuvent être utilisées en conjonction avec les attributs suivants :

- l'orientation et la forme des caractères,
- la justification et la taille des caractères,
- la taille de l'espace inter-caractères,
- l'orientation de la base et du plan des caractères,

4. Segmentation des images.

DI-3000 permet de former des images composées d'un ou plusieurs segments, chaque segment étant constitué du regroupement logique de plusieurs primitives graphiques.

La création d'un segment se déroule en trois étapes :

- l'ouverture d'un nouveau segment,
- la création de primitives graphiques,
- la fermeture du segment.

Les contraintes de segmentation imposent qu'une primitive appartienne a un et un seul segment. De plus, dès qu'un segment a été refermé, plus aucune primitive ne peut lui être ajoutée.

DI-3000 fournit deux types de segments, les segments retenus et les segments temporaires.

4.1. Les segments retenus.

Les segments retenus sont nommés et mémorisés dans une structure de données interne gérée par DI-3000 et appelée l'espace des segments (segment storage).

Un segment retenu est maintenu dans l'espace des segments jusqu'à ce qu'il soit détruit explicitement, ou implicitement lors de la terminaison de DI-3000.

Un segment retenu possède différents attributs statiques et dynamiques. A titre d'exemple, l'attribut de visibilité indique si un segment doit ou ne doit pas être rendu visible sur l'appareil sélectionné.

4.2. Les segments temporaires.

Les segments temporaires ne portent pas de nom et ne sont, en fait, qu'un mécanisme aisé pour grouper des primitives.

Il en découle que l'image d'un segment temporaire ne peut être effacée sélectivement. Toutefois, l'ensemble des segments temporaires peut être effacé de façon permanente par un changement d'image (new frame action).

Quel peut être l'intérêt d'une telle distinction ?
A titre d'illustration, notons que beaucoup d'applications non interactives manipulent des objets graphiques dont certaines caractéristiques ne varient pas d'une image à l'autre. C'est le cas d'un graphique pour lequel seules les données de la courbe peuvent changer. Dans cet exemple, les axes, les graduations et les annotations du graphique constituent le segment retenu, tandis que les courbes successives correspondent à des segments temporaires distincts.

5. Conversion, transformation et modelage d'images.

La conversion d'images établit la correspondance entre un objet décrit en coordonnées réelles et son image finale sur un appareil graphique, par l'intermédiaire du système de coordonnées virtuelles.

En deux dimensions, la conversion est entièrement définie par une fenêtre en coordonnées réelles et un champ dans le plan de vision (view surface). Ce champ est un rectangle spécifié en coordonnées virtuelles et ses côtés sont parallèles aux côtés horizontaux et verticaux du plan de vision. La fenêtre permet au programmeur de travailler dans un système de coordonnées qui sont naturelles pour son application. Elle peut être utilisée pour délimiter un objet (to clip), c'est-à-dire pour déterminer quelle partie de l'objet on veut voir apparaître dans la fenêtre.

En trois dimensions, la situation est un peu plus compliquée. Etant donné que nous n'avons pas abordé la représentation d'objets dans un tel système, nous nous bornerons à dire, ici, qu'en plus de la notion de fenêtre et de champ de vision, DI-3000 permet de définir un plan de projection et soit un centre de projection pour les projections en perspective, soit une direction de projection pour les projections parallèles.

La conversion d'images est à distinguer de la transformation et du modelage d'images (modelling transformation).

La transformation d'images permet la manipulation d'images en coordonnées virtuelles. Elle définit comment l'image d'un segment retenu peut être modifiée par des rotations, des translations et des changements d'échelle, sur la base du segment. La transformation s'applique à l'image d'un objet, après que la conversion réelle-virtuelle ait eu lieu.

Le modelage d'images permet la manipulation d'objets en coordonnées réelles, en trois dimensions. Le modelage définit comment l'objet peut être modifié par des rotations, des translations et des changements d'échelle avant que la "photographie" de l'objet ne soit prise. Plusieurs transformations spatiales peuvent lui être appliquées de façon individuelle ou composite.

6. Exemple d'application.

Le but de ce paragraphe est d'illustrer concrètement certaines notions introduites précédemment. Nous procéderons par étapes successives.

La figure 6 représente un simple système d'axes, sans graduations, les valeurs extrêmes des axes n'étant placées que dans le but de faciliter l'exposé.

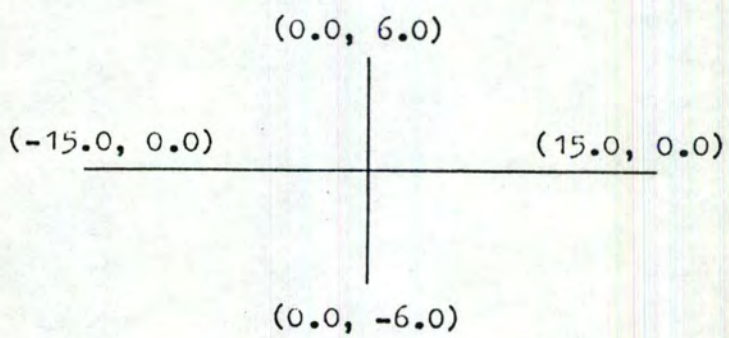


FIGURE 6
Système d'axes

Pour obtenir ce tracé dans un espace à deux dimensions, le programmeur doit définir les unités du système de coordonnées qu'il veut utiliser (système de coordonnées réelles).

Pour ce faire, il doit préciser la limite inférieure et supérieure de la fenêtre, dans l'espace des coordonnées réelles, qu'il veut voir affichée sur l'appareil graphique. Pour le cas considéré, il choisira par exemple une fenêtre s'étendant de -16 à 16 en abscisses et en ordonnées.

La figure 7 montre comment cette fenêtre (représentée en trait pointillé, à gauche) correspond au plan de vision (à droite) et illustre la façon dont les axes seront représentés dans ce plan de vision.

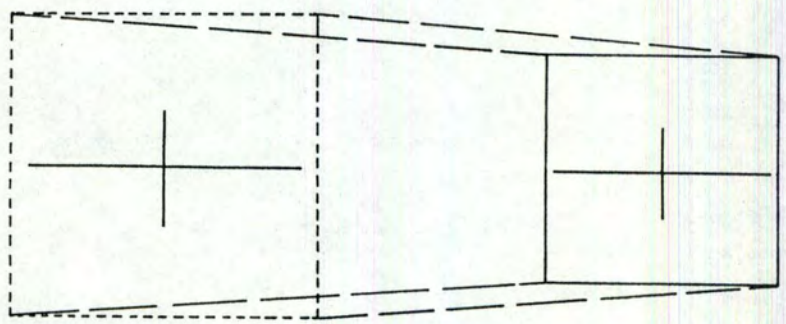


FIGURE 7
Fenêtre sans délimitation d'images

L'élimination d'une partie du système d'axes du plan de vision peut être obtenue en modifiant la taille de la fenêtre (celle-ci s'étend maintenant de -5 à 15 en abscisses et de -10 à 10 en ordonnées) et en autorisant la délimitation d'images (clipping). En effet, comme on peut le constater sur la figure 8 ci-dessous, la partie gauche du système d'axes n'apparaît plus dans le plan de vision.

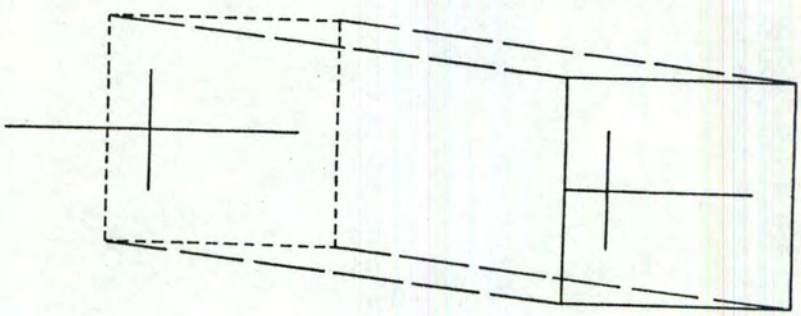


FIGURE 8
Fenêtre avec délimitation d'images

De plus, on obtient un effet d'agrandissement du dessin consécutif à la réduction de la taille de la fenêtre.

Jusqu'à présent, nous avons considéré que la fenêtre coïncidait avec le plan de vision tout entier. En réalité, il est possible de ne sélectionner qu'une région de ce plan de vision en définissant un champ de vision (viewport) dont les limites sont exprimées en coordonnées virtuelles.

La figure *9* illustre la façon dont s'établit la correspondance entre la fenêtre et le champ de vision par le mécanisme de la conversion d'images.

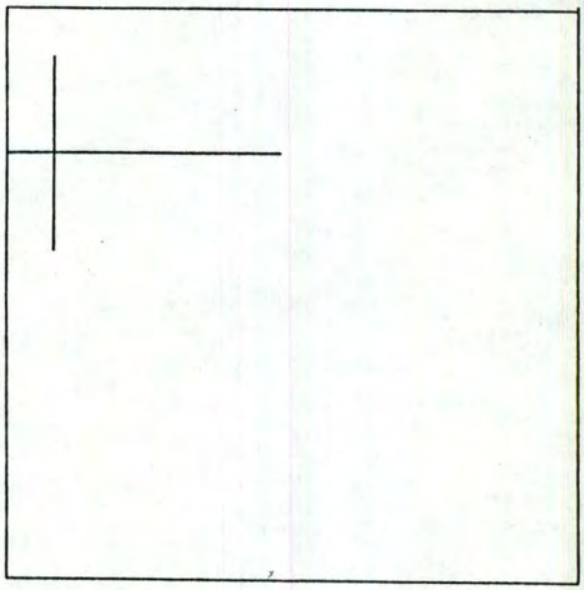


FIGURE 9
Champ de vision unique

Le champ de vision a été choisi comme s'étendant de -1 à 0 en abscisses et de 0 à 1 en ordonnées(quadrant superieur gauche). En sélectionnant successivement les autres quadrants, on peut obtenir le dessin de la figure 10 ci-dessous.

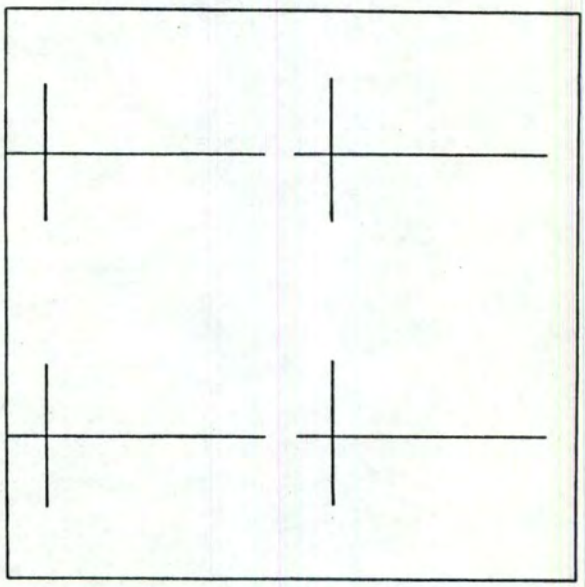


FIGURE 10
Champs de vision multiples

Il est à noter qu'il n'est possible de définir qu'un seul champ de vision à la fois.

Il faut également préciser que, dans le cas où le rapport des côtés de la fenêtre n'est pas égal à celui des côtés du champ de vision, la conversion d'images génère un graphique dont la forme est comprimée en ordonnées ou en abscisses.

Nous pouvons maintenant compléter notre graphique par l'addition d'une polyligne joignant un ensemble de points dont on fournit les coordonnées. Nous obtenons la figure **11** .

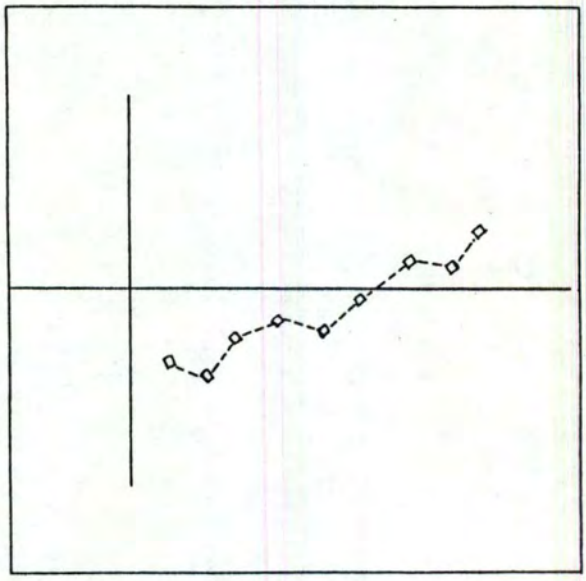


FIGURE 11
Polyligne et symboles spéciaux

Cette figure illustre également la notion de symboles spéciaux qui permettent la localisation de certains points de la polyligne. De plus, cette dernière a été tracée en utilisant un style de trait particulier (tirets).

Enfin, cet exemple d'application peut être annoté (graduations, nom des axes, nom du dessin, ...) en utilisant les primitives de texte, dont les attributs ont été définis précédemment.

2.2.2. Le métafichier DI-3000.

Il s'agit d'un fichier séquentiel d'images mutuellement exclusives. Une image est constituée de toutes les informations graphiques générées entre deux changements d'images (new frame action).

L'avantage d'un tel outil est double.

D'une part, il s'agit d'un moyen commode pour archiver les images résultant de l'exécution d'un programme graphique. On évite ainsi la répétition des calculs nécessaires à l'obtention de ces images.

D'autre part, le métafichier permet le transfert aisé des images d'un environnement graphique à un autre, étant donné la standardisation dont le format du métafichier a fait l'objet.

Le métafichier est créé à l'aide du générateur de métafichier.

A. Le générateur de métafichier.

Le programme d'application traite le métafichier comme s'il s'agissait de n'importe quel appareil graphique. Pour ce faire, il doit initialiser et selectionner un contrôleur particulier, le générateur de métafichier. Au lieu d'envoyer des informations graphiques vers un appareil physique, le générateur construit une librairie séquentielle d'images, le métafichier.

Ce dernier peut donc être considéré comme une trace des commandes graphiques qui autrement auraient été envoyées vers l'appareil physique.

Lorsque le métafichier a été créé, il peut être traité à l'aide d'un éditeur d'images appelé l'interpréteur de métafichier (metafile translator).

B. L'interpréteur de métafichier.

Il s'agit d'un programme interactif qui permet d'afficher sur un écran vidéo les images contenues dans un métafichier. L'interpréteur lit les images sélectionnées, les positionne et les met à l'échelle avant de les envoyer vers l'appareil.

Le fonctionnement de l'interpréteur est illustré à l'aide de la figure 12 .

L'interpréteur apparaît comme un programme indépendant qui s'exécute en relation avec le gérant du réseau DI-3000 et un contrôleur d'appareil sélectionné.

L'utilisateur lance l'interpréteur et lui fournit les commandes interactivement à partir d'un terminal. Ces commandes peuvent également être lues dans un fichier préalablement créé. La manipulation de l'interpréteur est décrite dans [DI], Appendice D.

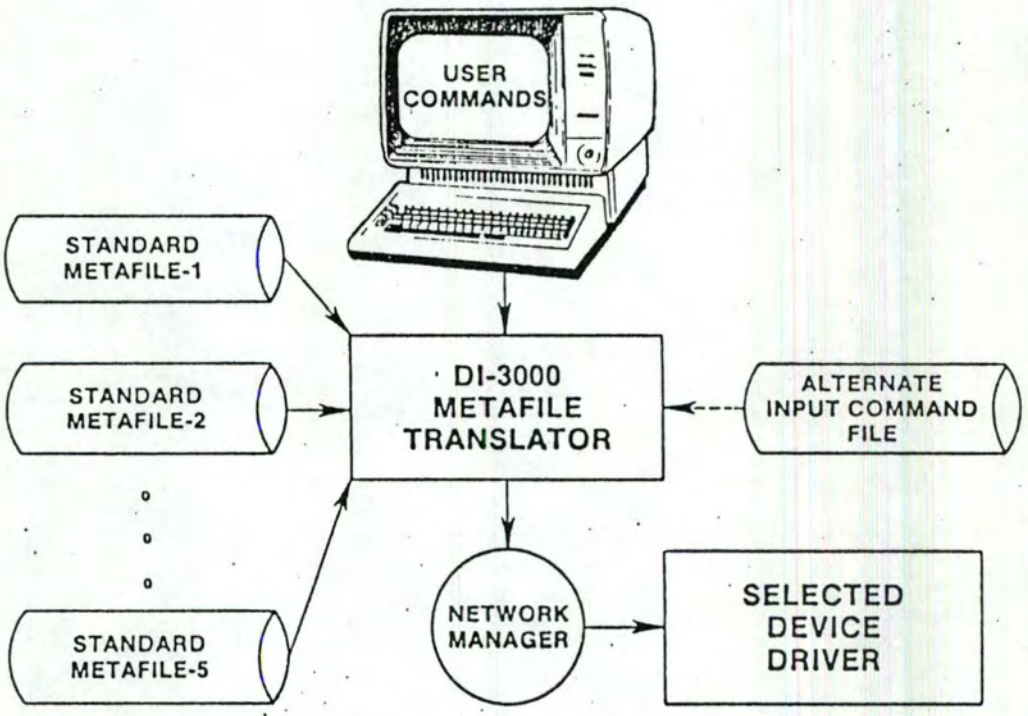


FIGURE 12

Fonctionnement de l'interpréteur

2.2.3 Glossaire - definitions fondamentales.

Ce glossaire reprend, en ordre alphabétique, la liste des termes communément utilisés dans la description des logiciels graphiques.

BATCH OF UPDATES (Mise à jour différée)

Une séquence de modifications d'image groupées par le programmeur d'application pour des raisons d'efficacité. Ainsi, les modifications successives sont mémorisées et ne sont pas représentées immédiatement sur l'appareil. Ce n'est qu'après un certain délai que l'image finale est obtenue. Ce mécanisme permet d'éviter les effaçages multiples consécutifs aux modifications d'image. Toutefois, l'effet de cette mise à jour différée reste fortement dépendante de l'appareil utilisé.

CLIPPING (Délimitation)

Elimination de toutes les primitives qui se trouvent en dehors de la fenêtre, de telle façon que seule la partie de l'objet qui se trouve à l'intérieur de la fenêtre est dessinée. La fenêtre peut alors être considérée comme le viseur d'une caméra, où seule la partie de l'espace qui est visible dans ce viseur est finalement fixée sur la pellicule.

DEVICE DRIVER (Contrôleur d'appareil)

Ensemble de sous-programmes qui interprète les commandes graphiques du niveau indépendant de DI-3000 et convertit ces commandes en des instructions spécifiques à un appareil physique. Le générateur de métafichier est un exemple particulier de contrôleur d'appareil.

DISPLAY SURFACE or VIEW SURFACE (Plan de vision)

La région physique d'un appareil (de dessin) sur laquelle une image est générée. Les dimensions de cette région varient d'un appareil à l'autre et sont exprimées en unités physiques. Par exemple, les dimensions communes d'un écran couleur à balayage sont :

(80 ≤ coord en X ≤ 560)
(0 ≤ coord en Y ≤ 480) en pixels

et les dimensions du métafichier DI-3000 sont :

(0 ≤ coord en X ≤ 32767)
(0 ≤ coord en Y ≤ 32767)
en unités sans dimensions.

NDC COORDINATES or NORMALIZED DEVICE COORDINATES

(Coordonnées virtuelles ou normalisées)

Le système de coordonnées d'un appareil graphique virtuel. Ces coordonnées varient toujours entre -1.0 et 1.0 en X et en Y.

NEW FRAME ACTION (Changement d'image)

Effaçage de la région d'affichage et retraçage, si nécessaire, de tous les segments retenus et visibles. Cette action peut être provoquée explicitement pour préparer, par exemple, une nouvelle image. Généralement, elle est l'effet secondaire de l'invocation d'un sous-programme DI-3000. En relation avec une table traçante, elle provoque l'avance du papier pour le dessin suivant.

SEGMENT STORAGE (Espace des segments)

Structure de données destinée à la mémorisation des segments retenus. Quand le premier segment retenu est créé, DI-3000 réserve une certaine quantité de mémoire pour cette structure. Cette mémoire est allouée à la demande chaque fois que des segments sont créés, et rendue à nouveau disponible lorsque des segments sont détruits.

VIRTUAL GRAPHICS DEVICE (Appareil graphique virtuel)

Appareil graphique idéalisé représentant l'union des capacités disponibles sur la plupart des appareils physiques. Le programmeur d'application écrit des programmes orientés vers cet appareil virtuel.

WORLD COORDINATES SYSTEM (Système de coordonnées réelles)

Le système de coordonnées à deux ou trois dimensions dans lequel le programmeur définit des objets. Il choisit les unités de ce système dans son programme d'application.

C H A P I T R E 3

LES SOLUTIONS.

3.1 La problématique du Pascal.

3.1.1 Introduction.

DI-3000 étant un logiciel écrit en Fortran, il était naturel d'utiliser ce même langage pour la partie programmation de ce mémoire. Toutefois, on constate actuellement une utilisation de plus en plus large du Pascal dans l'informatique, et notamment à l'Institut d'Informatique, où il a remplacé le Basic en tant que langage de base, et le Fortran pour les applications scientifiques.

Nous avons donc décidé de mettre les outils graphiques qui constituent ce mémoire également à la disposition des utilisateurs du Pascal.

Cette décision amène immédiatement la question suivante : peut-on appeler des procédures écrites en Fortran dans un programme d'application écrit en Pascal ? De manière plus générale, les deux langages sont-ils compatibles sur le DEC 20/60 dont nous disposons à Namur ?

La réponse est malheureusement négative. L'expérience montre en effet qu'il est possible de compiler un programme Pascal faisant appel à des procédures Fortran, mais que l'exécution d'un tel programme peut échouer pour des raisons indéterminées.

Nous n'avons pas pu déterminer avec précision l'origine de ce phénomène, mais voici néanmoins quelques éléments qui pourraient l'expliquer.

3.1.2 Incompatibilité logicielle.

Il existe sur le DEC deux types de langages.

1) Ceux du type Cobol, Fortran, Assembleur, dont les compilateurs sont d'origine DEC et qui respectent tous la même structure pour les fichiers exécutables (zone code objet, zone de travail, zone de données).

2) Ceux du type Pascal, qui ne sont pas d'origine DEC, et qui ne respectent donc pas nécessairement la même structure pour ces mêmes fichiers.

D'autre part, il existe pour chaque langage un ensemble de routines appelées " Object Time System " qui sont utilisées par le relieur et qui réalisent des points particuliers du système d'exploitation TOPS 20, tels que:

- la gestion des entrées-sorties,
- la gestion dynamique de mémoire,
- etc.

Or le Pascal ne semble pas avoir un " Object Time System " compatible avec ceux fournis par DEC.

La recherche d'une solution à ce problème d'incompatibilité, et donc l'étude précise du comportement d'un programme Pascal lors de son execution n'est pas envisagée pour l'instant au centre de calcul. Cela pour deux raisons : La première est que l'apparition d'un Pascal d'origine DEC rendrait inutile toute étude réalisée. Deuxièmement, nous sommes passés il y a quelques semaines de la version IV à la version V du Fortran et le centre de calcul a déjà prévu le passage aux versions VI et VII. Dès lors toute solution ne resterait valable qu'un certain temps.

Cette incompatibilité entre les deux langages a nécessité la séparation du mémoire en deux parties.

3.1.3 Le dédoublement du mémoire.

Notre objectif, qui est, rappelons le (cf introduction) de munir d'outils graphiques le Pascal autant que le Fortran, et l'incompatibilité entre ces deux lagages a deux conséquences :

- d'une part, il faut programmer dans les deux langages les solutions apportées aux problèmes constituant ce travail.

- d'autre part, il faut réécrire en Pascal un ensemble de procédures qui joueront le rôle d'outil de base tenu par DI-3000 pour la partie Fortran.

Une première possibilité pour la constitution de cet outil était la construction d'une librairie équivalente en tout point à DI-3000. Par manque de temps, cette solution a dû être abandonnée et nous avons été amené à limiter les objectifs de cette partie Pascal. Nous avons décidé de nous baser sur la notion de métafichier, et de construire une librairie de procédures dont l'effet est de tracer des dessins uniquement dans des métafichiers.

Rappelons que cela veut dire que les dessins créés ne sont pas visibles immédiatement à l'écran, car l'affichage du contenu d'un métafichier nécessite l'utilisation d'un programme spécialisé appelé interprèteur de métafichier, et qui est une partie du logiciel DI-3000.

Par contre, en ce qui concerne le tracé des dessins sur la table traçante, cela n'entraîne pas de différence par rapport à l'utilisation de DI-3000, puisque dans la configuration disponible au centre de calcul, il est prévu qu'un dessin sur la table doit se faire à partir d'un métafichier.

Une autre caractéristique des métafichiers est que le contenu d'un tel fichier ne peut être modifié partiellement. Dans ce cas, il faut en effet recréér entièrement le ou les dessins qu'il contenait.

Nous allons à présent aborder l'étude de cette librairie que nous avons appelée générateur Pascal de métafichier par analogie avec la notion de générateur de métafichier DI-3000 qui a été vue en 2.2.2 .

3.1.4 Le générateur Pascal de métafichier.

Ce générateur est une librairie de procédures appelables par des procédures ou des programmes de plus haut niveau.

Les procédures correspondent à des commandes élémentaires, qui sont regroupées en deux classes : les commandes de positionnement et celles de non-positionnement.

Les commandes de positionnement comprennent des ordres tels que :

- mouvement plume haute,
- mouvement plume basse.

Les commandes de non-positionnement comprennent des ordres tels que :

- fixation de la taille des caractères,
- fixation de l'espacement inter-caractères,
- fixation de la couleur des traits,
- fixation des attributs graphiques de polygones :
  - + couleur du bord,
  - + couleur de l'intérieur,
  - + style de l'intérieur (hachurage),
- délimitation du début d'un dessin,
- délimitation de la fin d'un dessin,
- délimitation du début d'un métafichier,
- délimitation de la fin d'un métafichier.

Toutes les informations complémentaires concernant la découpe en classes de la librairie, la liste complète des procédures, leurs spécifications et les ouvrages de référence se trouvent en annexe 1.

Nous allons à présent aborder l'etude détaillée des solutions apportées aux problèmes qui font l'objet de ce mémoire.

3.2 Présentation des solutions.

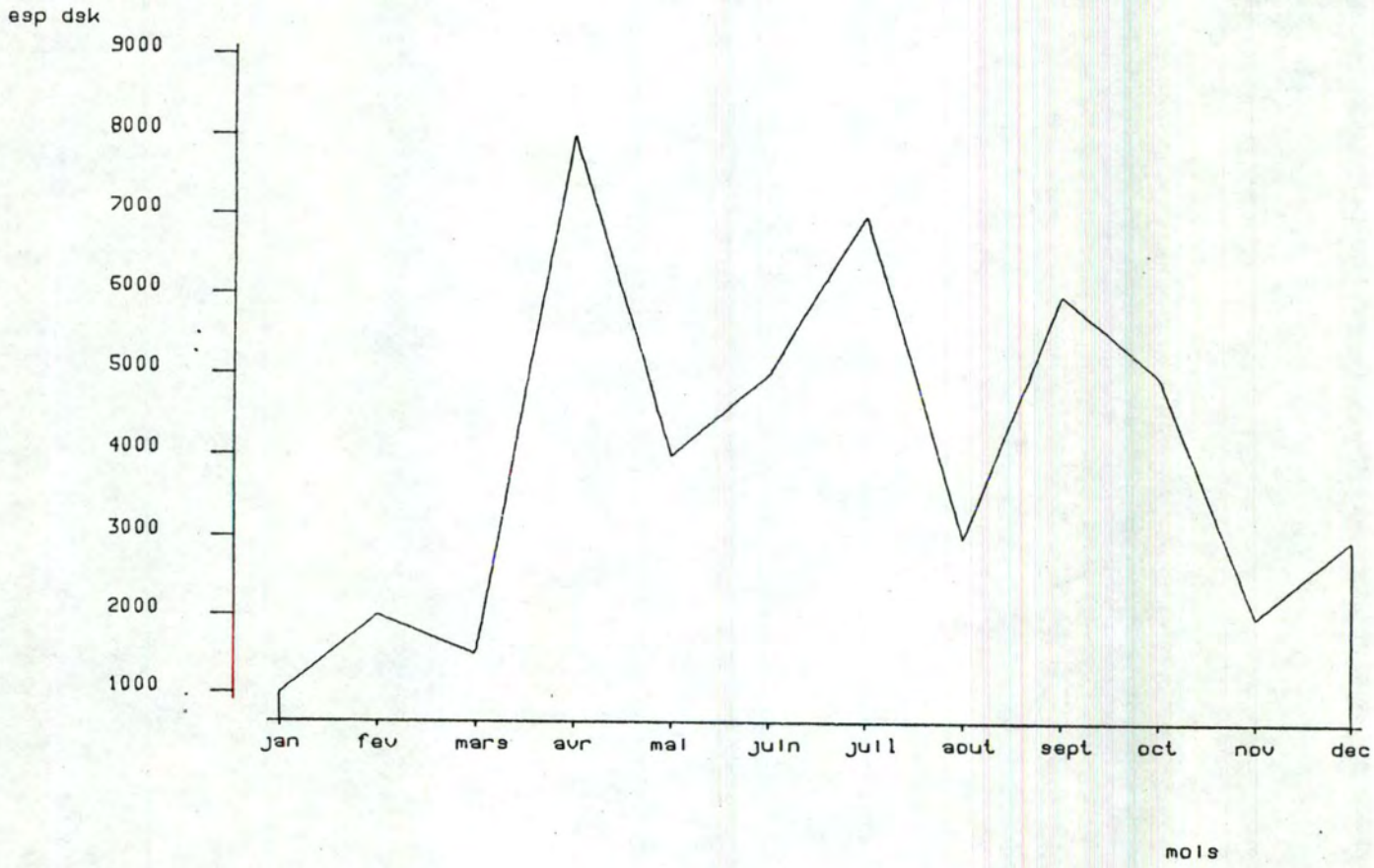
Il faut, dès à présent, faire une observation sur ce qui suit.

Vu le temps qui nous etait imparti, il nous a été impossible d'apporter une solution a tous les problèmes dans les deux langages, Pascal et Fortran.

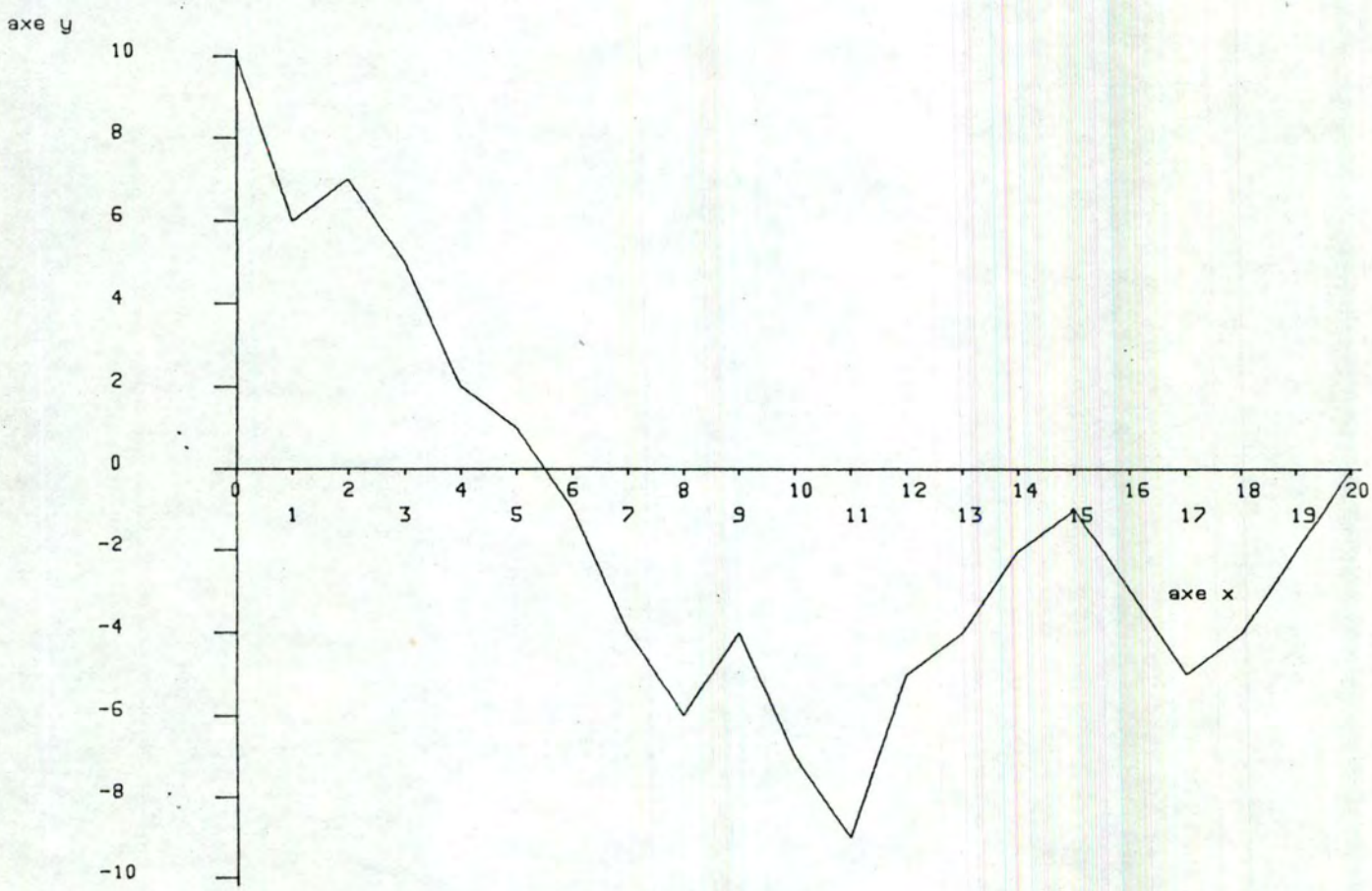
C'est ainsi que les solutions relatives aux fonctions d'une variable et à l'analyse en composantes principales ne sont disponibles qu'en Pascal. En ce qui concerne les graphes et les classifications hiérarchiques, seules les solutions en Fortran sont disponibles. Le problème des histogrammes a été traité, quant à lui, dans les deux langages.

3.2.1. Courbe x,y à deux dimensions.

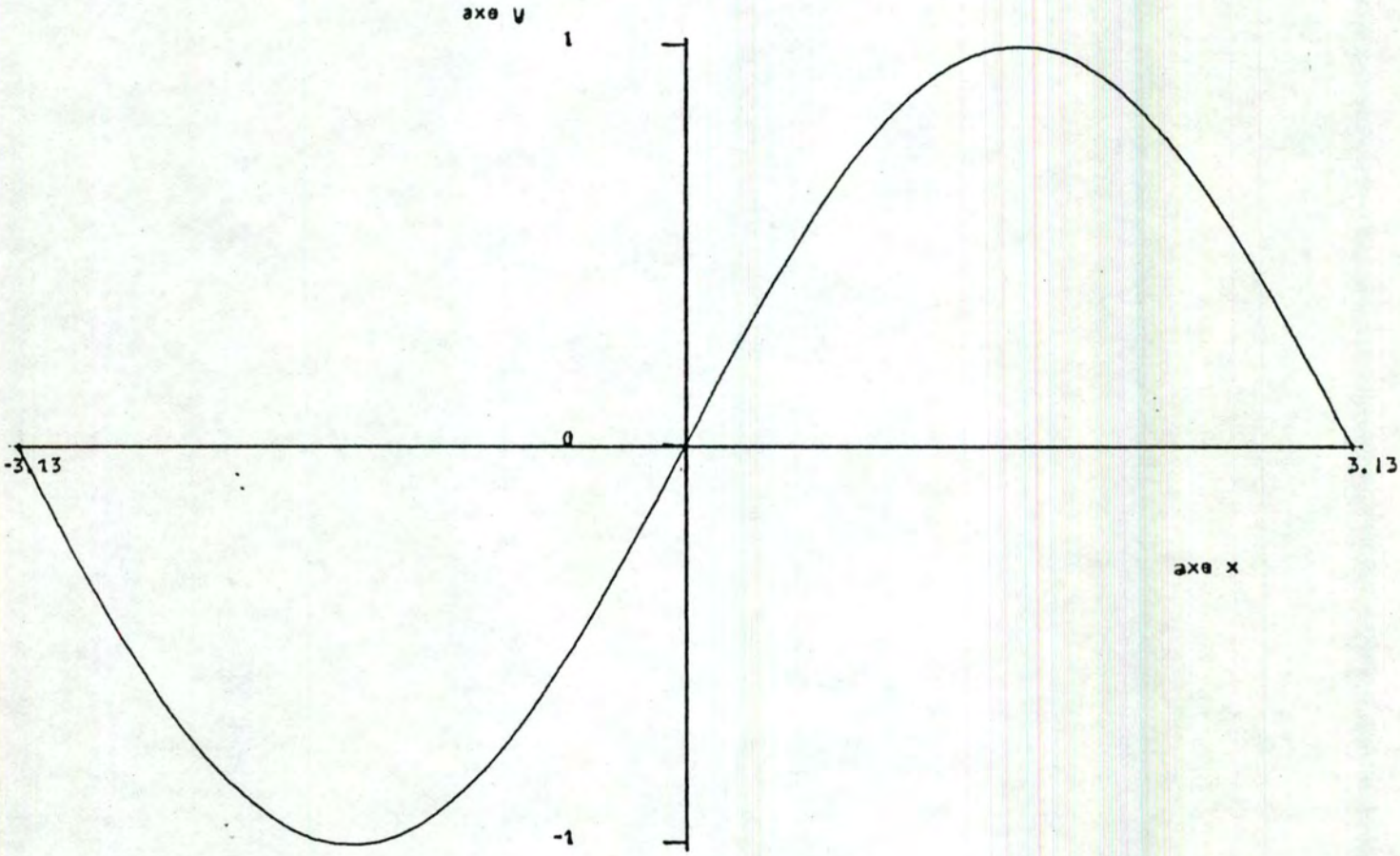
1. Exemples de réalisations.



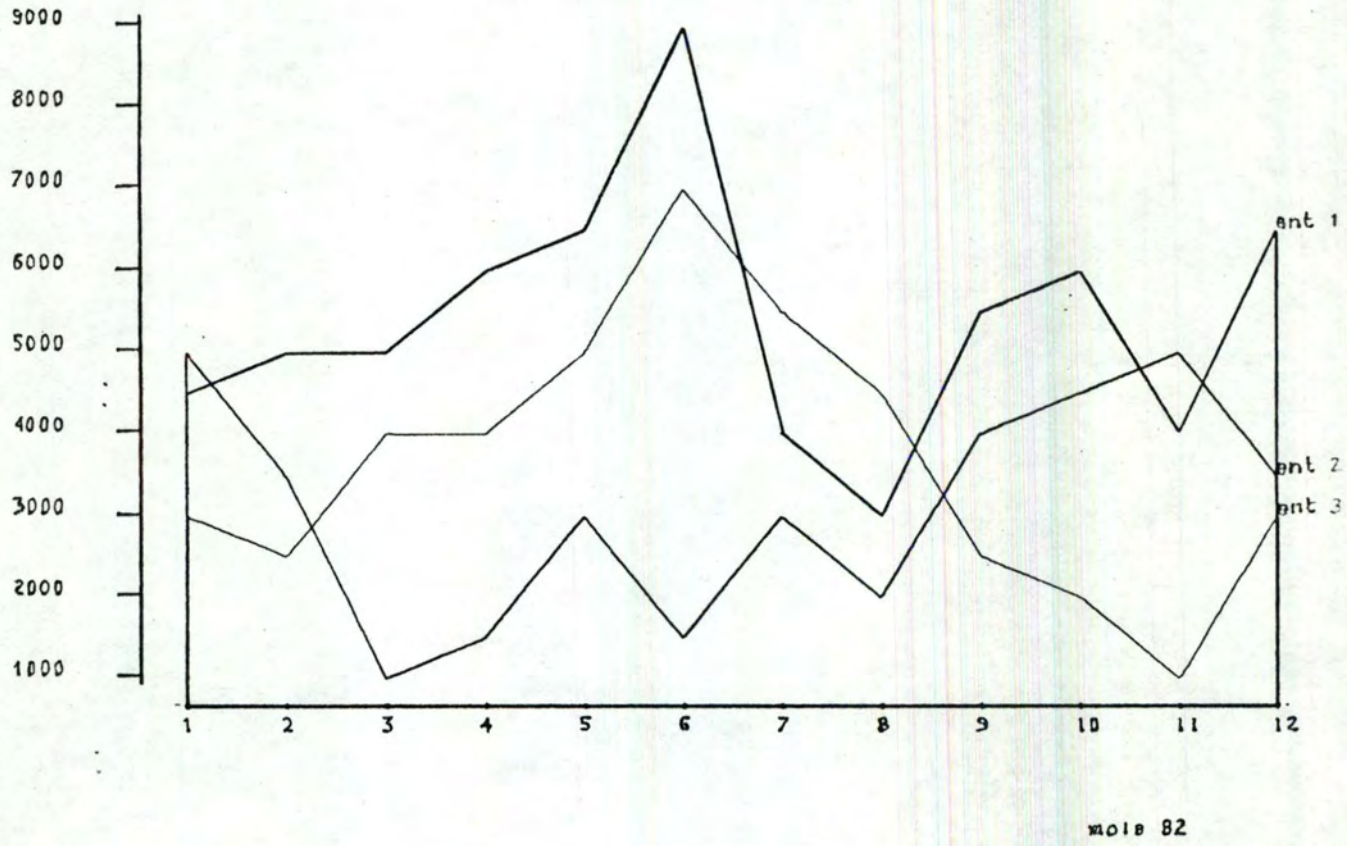
Exemple 1.


axe y
10
8
6
4
2
0
-2
-4
-6
-8
-10
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
axe x

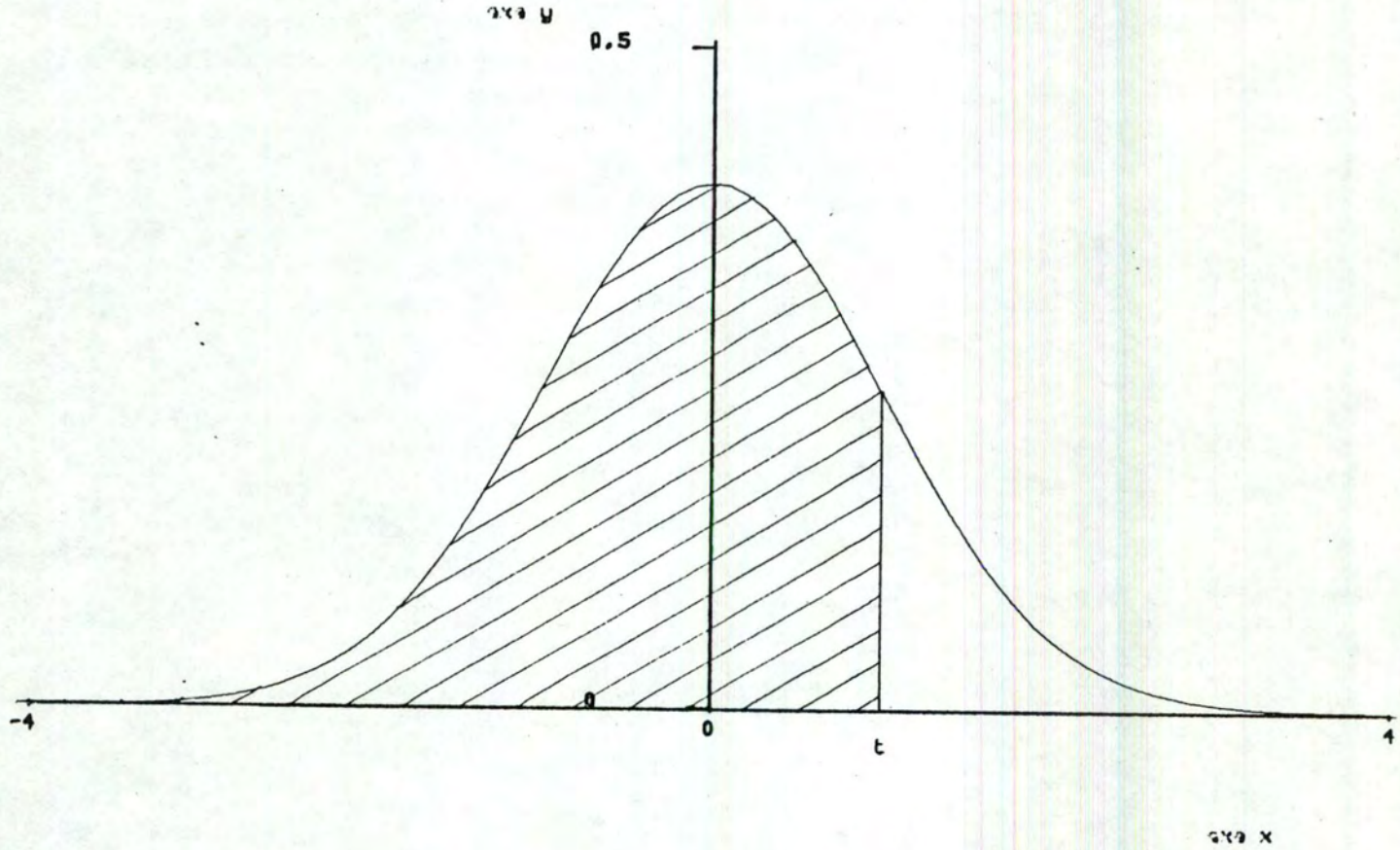
Exemple 2.



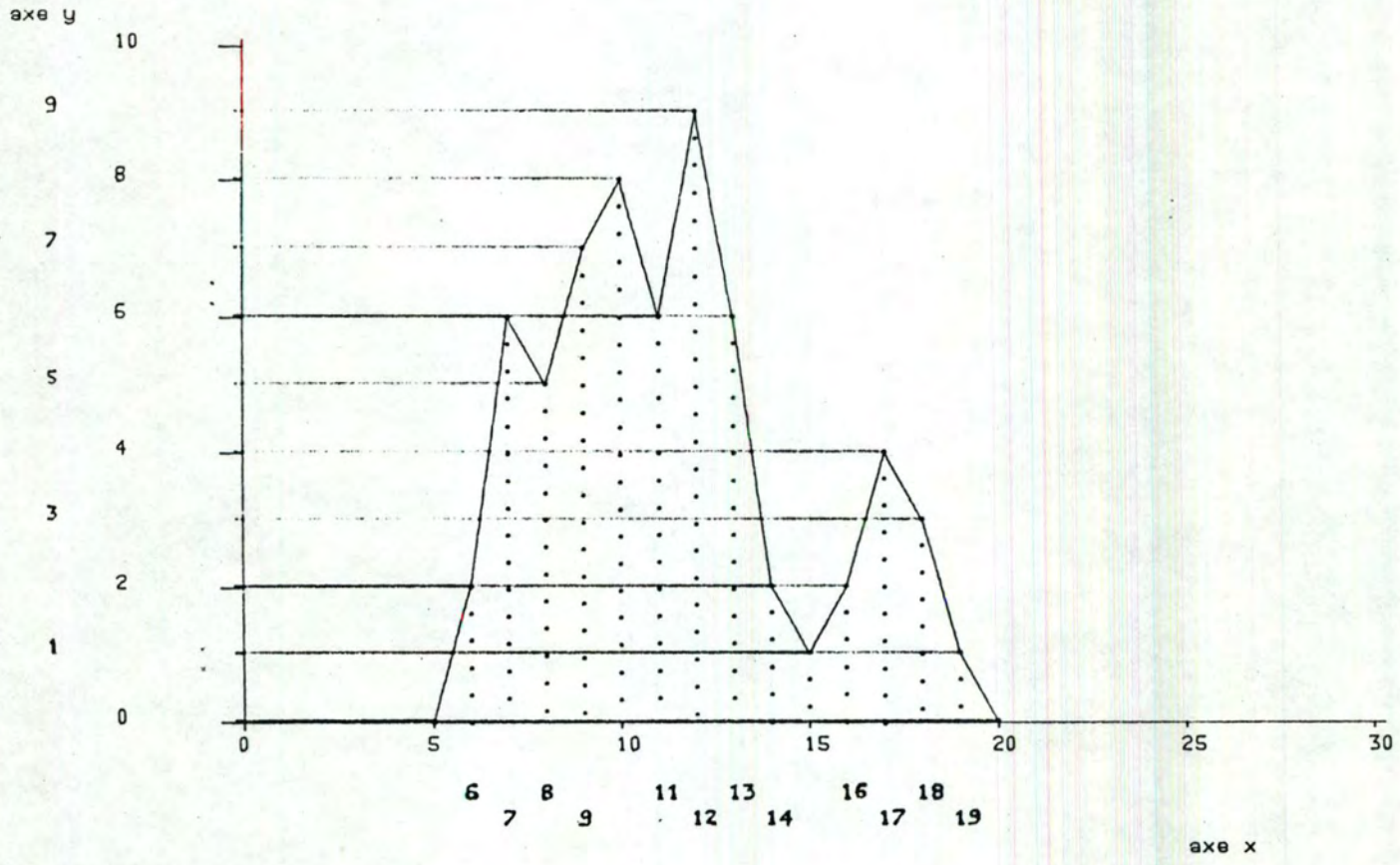
Exemple 3.


9000
8000
7000
6000
5000
4000
3000
2000
1000
1
2
3
4
5
6
7
8
9
10
11
12
ent 1
ent 2
ent 3
mois 82

Exemple 4.



Exemple 5.


axe y
10
9
8
7
6
5
4
3
2
1
0
0
5
10
15
20
25
30
6 8 11 13 16 18
7 9 12 14 17 19
axe x

Exemple 6.

La création de ces dessins est basée sur l'utilisation de la procédure graphique FCT dans le programme d'application écrit par l'utilisateur. Les données que ce dernier doit fournir à cette procédure sont principalement, d'une part, les valeurs de graduations des axes, et d'autre part les valeurs des coordonnées des points par lesquels doit passer la (les) courbe(s).

Nous allons donc passer à l'étude de FCT. Plus précisément, nous allons aborder les points suivants :

- les possibilités offertes à l'utilisateur de la procédure FCT;
- les contraintes imposées aux données, à l'entrée de FCT;
- la justification de ces contraintes.

Les détails concernant l'utilisation de la procédure se trouvent en annexe A .

2. La procédure FCT.

La procédure FCT est constituée de quatre parties :

- la vérification des contraintes,
- le tracé du système d'axes,
- le tracé de la courbe,
- le tracé de la projection de points de la courbe sur les deux axes.

Nous allons parcourir ces différents points dans l'ordre.

2.1. Les contraintes.

Chaque donnée fournie à FCT doit respecter des contraintes qui sont vérifiées au début de la procédure. Lors de la détection d'une erreur, deux cas sont envisagés :

- si l'erreur est susceptible de rattrapage, la variable en cause est remise à une valeur par défaut prédéterminée, et l'exécution de FCT se poursuit normalement;

- si l'erreur est plus grave, un message d'erreur apparaît et l'exécution de FCT est définitivement interrompue.

La liste complète des erreurs et de leur traitement sera décrite au fur et à mesure de l'exposé.

2.2. Le tracé du système d'axes.

Comme nous l'avons déjà dit plus haut, pour obtenir le tracé des axes, il faut fournir à la procédure les valeurs des graduations de chaque axe. Mais par ailleurs, l'utilisateur a la possibilité de choisir la couleur et le style utilisés pour la représentation des axes et le nom de chaque axe.

a) Les graduations.

Les valeurs des graduations peuvent être numériques ou alphanumériques.

a.1. Les valeurs numériques.

Ce sont des nombres réels qui doivent respecter des contraintes, tant au niveau individuel qu'au niveau de l'ensemble des valeurs.

Au niveau individuel, chaque valeur ne peut comprendre plus de cinq caractères. Les nombres limites sont donc $\pm$ 9999, le signe et le point décimal valant chacun un caractère. (Le + est optionnel et est remplacé par un blanc).
Deux raisons ont amené cette décision quant au format :

- une taille supérieure diminue le nombre de graduations possibles le long de l'axe et nuit à la lisibilité,
- en Fortran, une chaîne de cinq caratères peut être stockée dans une seule variable simple précision.

Si une graduation dépasse les valeurs limites, un blanc la remplace sur le dessin.

L'ensemble des valeurs doit respecter les contraintes suivantes :

* Pour des raisons de cohérence, la suite des graduations doit être strictement croissante. Si ce n'est pas le cas, l'exécution de FCT s'interrompt après l'envoi d'un message d'erreur.
* Un axe peut n'avoir que des graduations strictement positives ou négatives (exemple 1). Mais, lorsque l'ensemble des graduations comprend des valeurs positives et négatives, la valeur 0 doit être explicitement comprise dans cet ensemble. Ceci permet de déterminer la position de l'intersection des deux axes. Le non-respect de cette contrainte provoque l'apparition d'un message d'erreur et l'arrêt de FCT.
* Le nombre n des graduations doit être compris entre 2 et 50. En effet, on ne peut plus parler d'un axe si n 2, tandis que n 50 entraine un problème de lisibilité dû aux recouvrements. Le non-respect de cette contrainte provoque l'apparition d'un message d'erreur et l'arrêt de FCT.

a.2. Les valeurs alphanumériques.

Ce sont des chaînes de caractères qui doivent respecter les règles suivantes :

- Pour des raisons de lisibilité déjà évoquées ci-dessus, seuls les cinq premiers caractères de chaque graduation sont écrits le long de l'axe. (cf a.1.);
- Le nombre des graduations doit être compris entre 2 et 50; (cf a.1.);
- Seul l'axe des X (celui des abscisses) peut être alphanumérique (un axe (alpha)numérique est un axe dont les valeurs des graduations sont (alpha)numériques).

b) La couleur.

Il existe sept couleurs différentes. Le noir, qui est la huitième couleur disponible pour un terminal du type GIGI, n'a pas été repris dans la liste des couleurs possibles. En effet, le noir est utilisé comme couleur de fond lors de l'affichage d'un dessin à l'écran par l'interpréteur de métafichier.

L'utilisateur doit choisir une des sept couleurs prévues. S'il ne le fait pas, le tracé est exécuté en blanc, qui est la couleur par défaut. Il nous semble en effet que la non-indication d'une couleur constitue une erreur bénigne, pour laquelle l'arrêt de la procédure serait une sanction disproportionnée.

c) Le style.

Quatre styles de tracé sont disponibles. Le style par défaut est le trait continu. Notons que l'on choisit un style et une couleur pour le tracé des deux axes.

d) Le nom.

Chaque axe peut avoir un nom de dix caractères au maximum.

Pour clôturer cette étude des axes, signalons (voir l'exemple 6) que l'affichage des graduations se fait sur deux étages lorsque celles-ci deviennent trop nombreuses. D'autre part, les graduations numériques sont toujours représentées linéairement sur les axes.

Nous pouvons à présent passer à l'étude de la courbe.

2.3. Le tracé de la courbe.

Nous allons étudier la courbe sous deux aspects : d'abord en tant qu'ensemble de points, puis en tant qu'ensemble de zones.

A. La courbe en tant qu'ensemble de points.

Pour obtenir le tracé d'une courbe, l'utilisateur doit fournir à la procédure des points par lesquels la courbe doit passer. Chaque point possède une abscisse et une ordonnée dans le système de coordonnées défini par les axes que l'on a choisis.

Tout comme les axes, les coordonnées de chaque point peuvent être soit numériques soit alphanumériques.

a.1. Les coordonnées numériques.

Ce sont des valeurs réelles qui doivent respecter les règles suivantes.

* Le nombre de points définissant la courbe doit être compris entre 2 et 500. Etant donné la résolution des appareils graphiques, 500 points se sont révélés suffisants pour obtenir une finesse du tracé acceptable. Le non-respect de cette contrainte provoque l'apparition d'un message d'erreur et l'arrêt de FCT.
* Toutes les coordonnées numériques doivent avoir une valeur comprise entre la première et la dernière graduation de l'axe correspondant. Ceci assure que le tracé reste cohérent par rapport aux axes de référence. Si une coordonnée ne respecte pas cette contrainte, sa valeur est remplacée par celle de la graduation-limite la plus proche (en anglais : "clipping") et un message signalant le fait apparaît (l'exécution de FCT n'est pas interrompue).

a.2. Les coordonnées alphanumériques.

Nous avons déjà vu (cf 3.2.1.§ 2.2 point a.2) que seul l'axe X peut être alphanumérique. Dans un tel cas, on considère que chaque valeur des graduations de l'axe des X est l'abscisse d'un point de la courbe. De par ce fait, la procédure FCT considère que le nombre de points de la fonction est égal au nombre de graduations de l'axe X.

B. La courbe en tant qu'ensemble de zones.

Il est intéressant, dans certains cas, de pouvoir différencier certaines parties d'un dessin. Par exemple, on peut vouloir mettre en évidence une partie de l'aire comprise sous une courbe de densité, comme dans l'exemple 5 , où l'aire hachurée représente la valeur de la fonction de répartition en t.

Pour obtenir ce résultat, nous nous sommes basés sur une primitive du générateur de métafichier qui permet de définir sept hachurages différents pour un polygone (en plus de l'intérieur vide).

Nous sommes ainsi arrivé à la notion de zones.

b.1. Définition.

Une zone est l'aire délimitée par l'axe X, la courbe, et deux droites d'équation x = x1 et x = y1, x1 et y1 étant les abscisses de deux points distincts de la courbe. Par défaut, la procédure considère que les deux points sont le premier et le dernier de la courbe, ce qui veut dire que le dessin comprendra toujours au moins une zone.

C'est cette zone initiale que l'utilisateur a la possibilité de partitionner.

b.2. Caractéristiques d'une zone.

Graphiquement, une zone correspond à un polygone. L'utilisateur peut donc choisir pour ces derniers les attributs suivants :

1* La couleur du bord.

Sept couleurs sont possibles, le blanc étant la couleur par défaut.

2* Le style du bord.

Quatre styles sont possibles, le trait plein (continu) étant le style par défaut.

3* Le type de l'intérieur.

Ce paramètre permet d'obtenir un intérieur vide ou non vide. Dans le cas où l'on choisit la deuxième possibilité, les attributs 4 et 5 (voir ci-dessous) sont utilisés; sinon ils sont ignorés. Par défaut, l'intérieur est vide.

4* La couleur de l'intérieur.

Sept couleurs sont possibles, le blanc étant la couleur par défaut.

5* Le style de l'intérieur.

Sept hachurages différents sont disponibles, le style par défaut consistant en une série de lignes verticales.

6* Le nombre de points de la courbe repris dans la zone.

La procédure FCT se base sur cette donnée pour réaliser la découpe de la zone initiale. Le nombre doit bien entendu être compris entre 2 et 500, sinon un message d'erreur apparaît et FCT s'interrompt.

7* Le nom.

Chaque zone peut être qualifiée par une chaîne de dix caractères au maximum.

Enfin, avant d'aborder l'étude de la projection des points de la courbe sur les axes, signalons deux contraintes globales relatives aux zones que l'on peut définir :

- Le nombre de zones définies ne peut dépasser 40, pour une question de lisibilité. En cas de non respect de cette règle, la procédure reprend la valeur par défaut, qui est une seule zone recouvrant toute la courbe (cf b.1.).
- L'ensemble des zones définies doit former une partition de la zone unique initiale. Si ce n'est pas le cas, la découpe en zones est ignorée et, comme ci-dessus, une seule zone est dessinée.

2.4._ La_projection de_points de_la courbe_sur_les_deux axes.

L'utilisateur peut choisir les points de la courbe qu'il veut voir projeter sur les deux axes. Comme on peut le voir sur les exemples, les valeurs des abscisses (ordonnées) des points à projeter sont écrits le long de l'axe X (Y) - sauf si la valeur est celle d'une graduation de l'axe - sur une ligne (colonne) différente de celle des graduations des axes.

Comme dans le cas des graduations, un affichage à deux niveaux a été prévu pour l'axe X.

Le choix des points doit respecter la contrainte suivante :

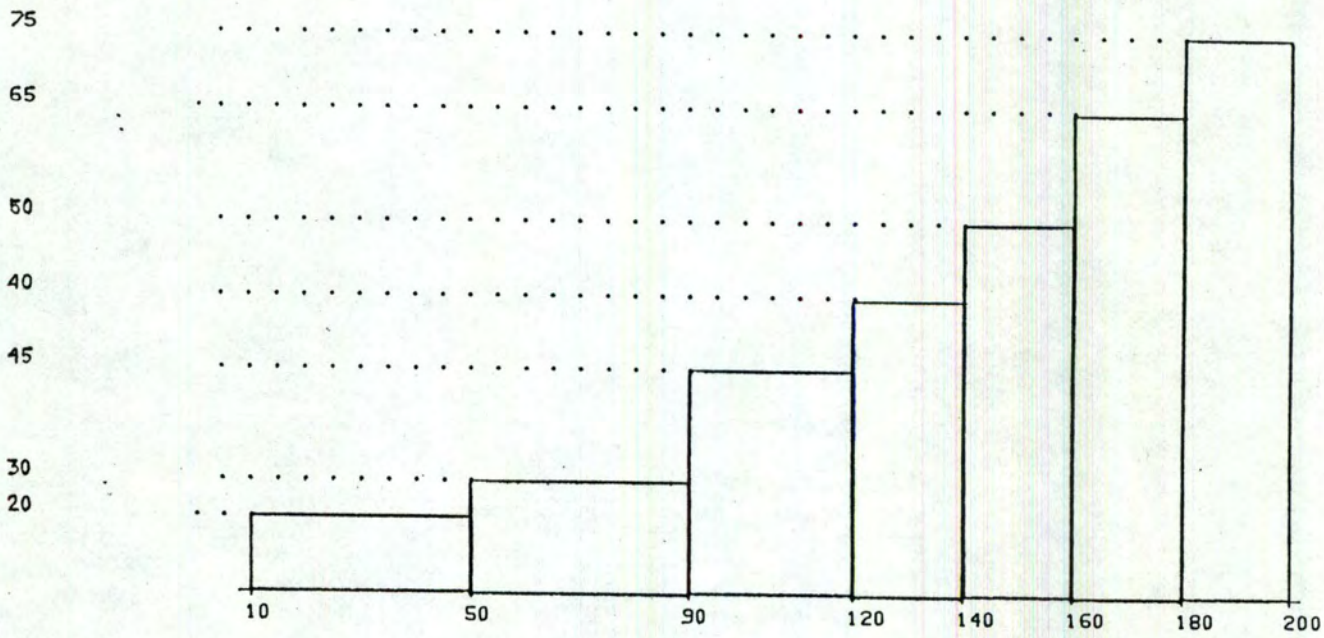
- Pour des raisons de lisibilité, le nombre des points à projeter doit être compris entre 0 et 50. Si cette règle n'est pas vérifiée, on ne projette aucun point.

Pour terminer notre étude de FCT, voyons une caractéristique générale du dessin : la rotation du dessin.

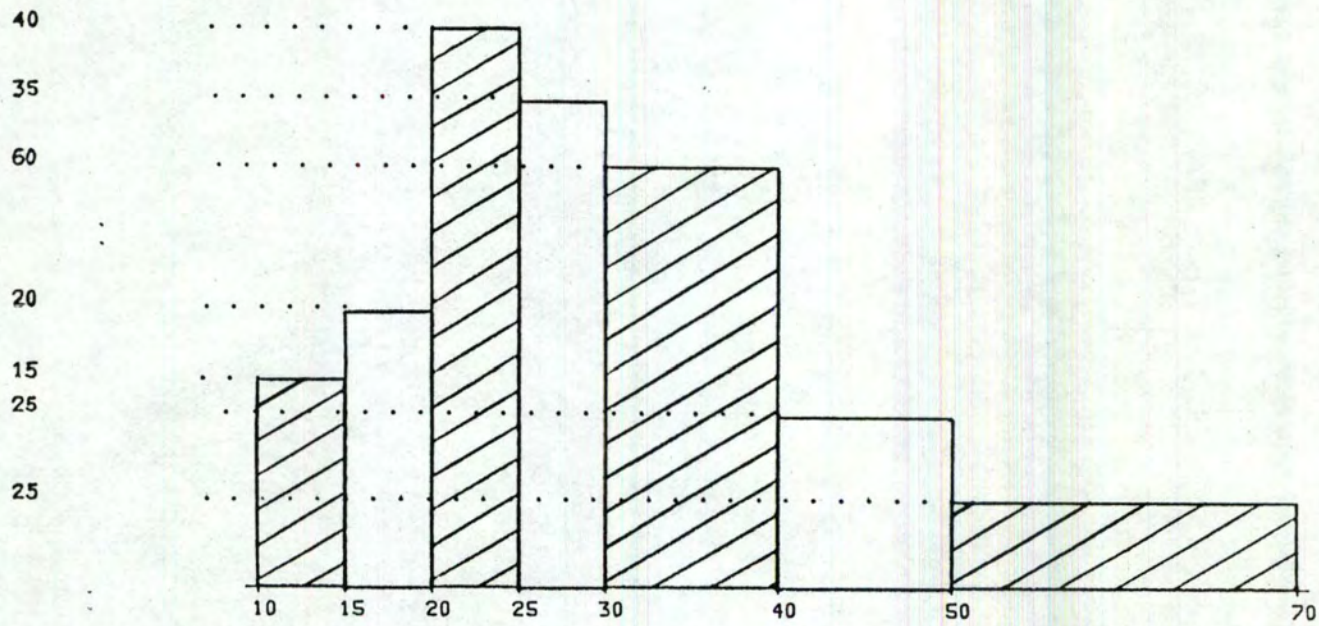
Lorsque le nombre des graduations de l'axe Y est plus grand que celui de l'axe X, le dessin entier généré par le (ou les) appel(s) à FCT subit une rotation de 90 degrés dans le sens des aiguilles d'une montre. Cela permet de profiter au maximum de la surface utilisable sur un moniteur vidéo.

3.2.2 Histogrammes.

1. Exemples de réalisations.



Exemple 1.



Exemple 2.

La création de ces dessins est basée sur l'utilisation de la procédure graphique HISTO dans le programme d'application écrit par l'utilisateur. Les données que ce dernier doit fournir à cette procédure sont principalement, d'une part, les valeurs des limites de classe, et d'autre part les valeurs des répétitions dans chaque classe.

Nous allons donc à présent passer à l'étude de cette procédure. Plus précisément, nous allons aborder les points suivants:

- les possibilités offertes à l'utilisateur de la procédure,
- les contraintes imposées aux données à l'entrée de la procédure,
- la justification de ces contraintes.

Les détails concernant l'utilisation de la procédure se trouvent en annexe A .

2. La procédure HISTO.

La procédure HISTO est constituée de quatre parties:

- la vérification des contraintes,
- le dessin de l'axe,
- le dessin des rectangles,
- l'affichage des répétitions pour chaque classe.

2.1 Les contraintes.

Le principe du traitement des erreurs est identique à celui de la procédure FCT (cf 3.2.1 § 2.1).

2.2 Le dessin de l'axe.

Comme nous l'avons déjà dit plus haut, pour obtenir le dessin de l'axe, il faut fournir à la procédure les valeurs des graduations de l'axe, qui sont ici les limites des classes de l'histogramme. Mais par ailleurs, l'utilisateur a la possibilité de choisir une couleur, un style et un nom pour l'axe.

a. Les graduations.

Les graduations sont des nombres réels. A nouveau, nous pourrions reprendre ici l'étude faite en 3.2.1., § 2.2, point a.1. Nous nous bornerons simplement à signaler deux différences :

1. La valeur 0 n'est plus une graduation obligatoire dans le cas où l'ensemble de ces dernières passe de valeurs négatives à des valeurs positives. En effet, cette contrainte ne se justifiait que par l'existence du système d'axes pour FCT. Or, rappelons qu'un histogramme ne comporte qu'un seul axe.

2. Le nombre des graduations doit être compris entre 2 et 41 et non plus 50 dans FCT. Le changement trouve son origine dans l'utilisation dans HISTO de sous-procédures qui avaient été écrites pour FCT.

b. La couleur, le style, le nom.

Ce qui a été dit à ce sujet lors de l'étude de la procédure FCT (cf. 3.2.1. § 2.2., points b), c) et d)) reste valable ici pour l'axe (unique) du dessin.

2.3. Le dessin des rectangles.

Les rectangles sont dessinés à partir des répétitions. Ces répétitions sont des nombres entiers qui doivent être positifs par définition. Lorsqu'une erreur est détectée, un message apparaît et la procédure s'interrompt.

Le nombre des rectangles est calculé par la procédure à partir du nombre des limites de classe.

Les caractéristiques de chaque rectangle sont identiques à celles des zones que nous avons vues en 3.2.1. § 2.3., point b.2. En effet, il s'agit dans les deux cas de polygones.

Nous citerons donc ici pour mémoire :

- la couleur du bord,
- le style du bord,
- le type de l'intérieur,
- la couleur de l'intérieur,
- le style de l'intérieur,
- le nom.

Remarquons que l'attribut "nombre de points de la courbe repris dans la zone" est ignoré ici, puisque ce sont toujours des rectangles qui sont tracés.

2.4. L'affichage des répétitions pour chaque classe.

Comme on peut le voir sur l'exemple **1** , il s'agit d'un recopiage des valeurs des répétitions à gauche du dessin, ce qui facilite l'interprétation du dessin.

Nous pouvons à présent passer à l'étude de l'analyse en composantes principales.

3.2.3. Analyse en Composantes Principales.

1. Rappels théoriques.

Avant de voir la solution proposée pour ce problème, nous allons faire un bref rappel de théorie. Ce rappel est tiré de [BB] , chapitre VII ou le lecteur peut trouver tous les compléments nécessaires.

L'analyse en composantes principales se base sur un tableau à deux entrées $T = (t_{ij} ; i = 1, \ldots, n \text{ et } j = 1, \ldots, p)$ représentant les valeurs prises par p variables quantitatives (par exemple : nombre de voitures par 1000 habitants, produit national brut, population totale, etc.) sur une population de n individus (par exemple n pays). Si l'on considère chaque individu i comme un vecteur de $\mathbb{R}^p$, dont les composantes sont les éléments de la $i^{ème}$ ligne du tableau T, on est en présence d'un nuage de points analogue à un ballon de rugby dans un espace à p dimensions.

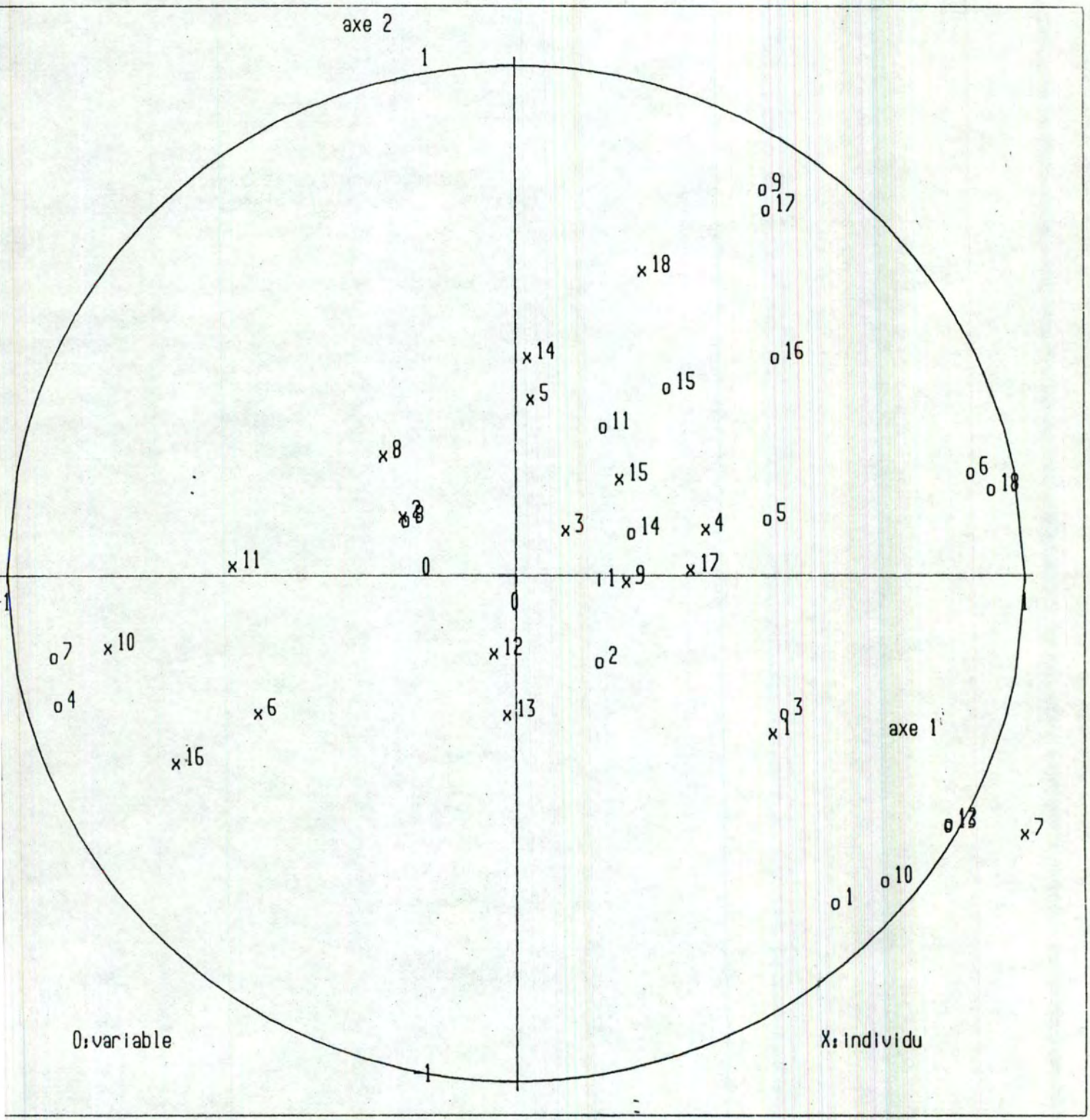
L'analyse en composantes principales se propose de saisir et comprendre la dispersion des points de façon progressive, c'est-à-dire en essayant d'abord de comprendre ce que signifie la dimension "allongement maximum du ballon", ensuite la dimension perpendiculaire à la précédente et prenant en compte le plus de dispersion possible, etc. Chaque dimension découverte expliquant une part de la dispersion, on s'arrête lorsqu'on a expliqué une part suffisante de celle-ci.

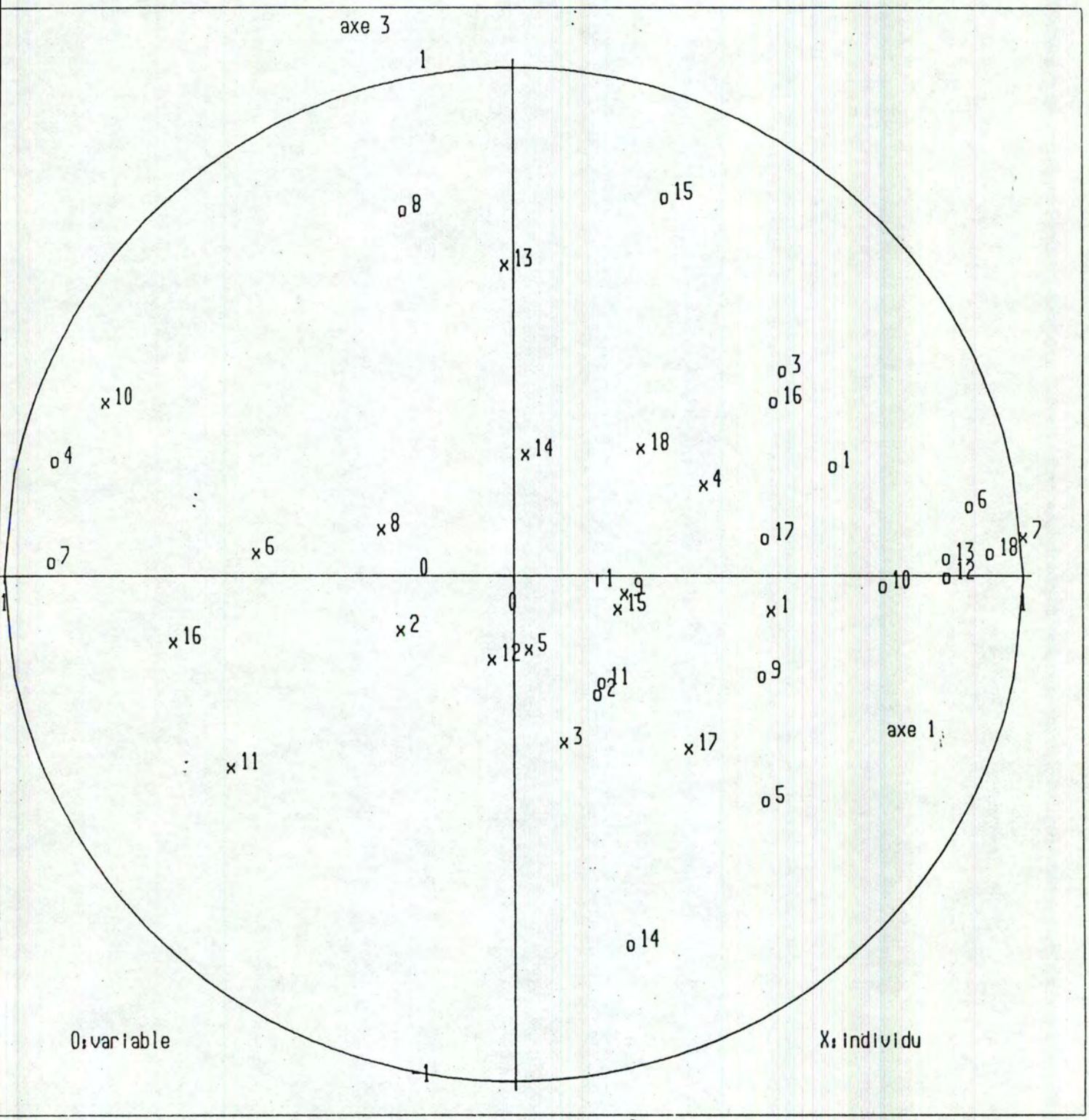
L'un des buts essentiels de l'analyse en composantes principales étant de fournir une représentation imagée de l'information contenue dans le tableau T, pour qu'elle soit plus propice à l'interprétation, on a évidemment intérêt à lui associer une procédure graphique de représentation. Dans ce qui suit, la procédure graphique ACP permet de visualiser :

a) tous les individus dans les axes factoriels, pris deux à deux, d'une analyse en composantes principales dans l'espace $\mathbb{R}^p$ des variables;

b) toutes les variables par rapport aux axes factoriels, pris deux à deux, d'une analyse en composantes principales dans l'espace $\mathbb{R}^n$ des individus, chaque variable j étant assimilée à un vecteur de $\mathbb{R}^n$ dont les composantes sont les éléments de la $j^{ème}$ colonne de T. Dans ce dernier cas cependant, nous avons préféré représenter les variables dans un système de coordonnées correspondant aux corrélations empiriques entre les variables et les axes factoriels de l'analyse dans $\mathbb{R}^n$.

2. Exemples de réalisations.


axe 2
axe 1
O:variable
X:individu


axe 3
axe 1
O: variable
X: individu

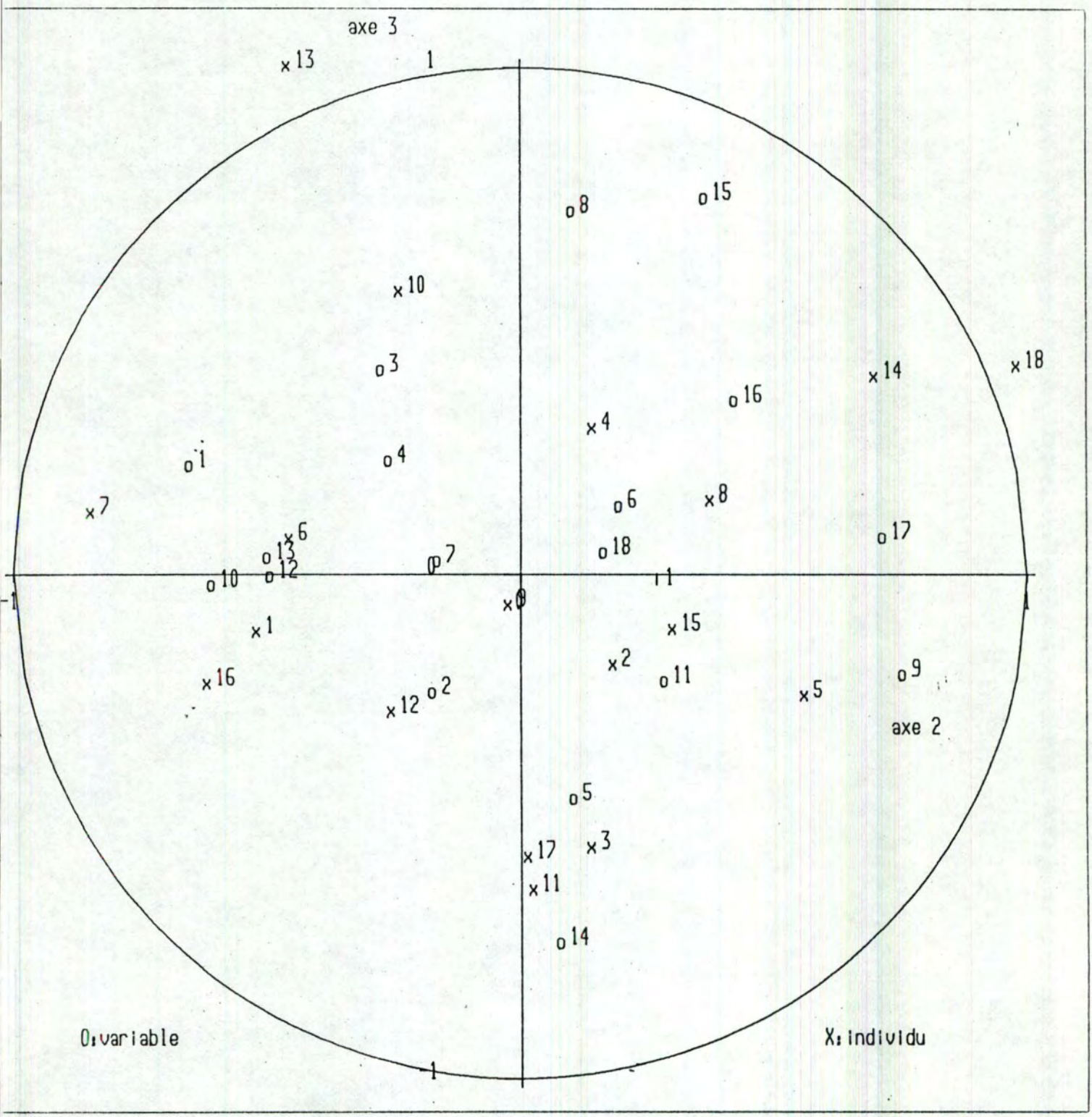

axe 3
axe 2
O: variable
X: individu

TABLEAU I. — *Code des caractéristiques et des pays*

Caractéristiques (variables)		*Nos*	*Pays (individus)*	
Population totale (1 000 habitants)	*POPU*	1	Allemagne	D
Densité (au km²)	*DENS*	2	Autriche	A
Taux d'accroissement total de la population	*TATO*	3	Belgique - Luxembourg	B
% d'actifs dans l'agriculture, la sylviculture et la pêche	*AASP*	4	Canada	CDN
% d'actifs dans l'industrie	*AIND*	5	Danemark	DK
Produit National Brut en dollars par habitant	*PNB*	6	Espagne	E
% du P.I.B. en agriculture	*PIBA*	7	Etats-Unis	USA
Formation brute du capital fixe en % du P.N.B.	*FBCF*	8	Finlande	SF
Recettes courantes en % du P.N.B.	*RECC*	9	France	F
Réserves officielles (en millions de dollars)	*RESO*	10	Grèce	G
Taux d'escompte officiel	*TESC*	11	Irlande	SE
Importations marchandises totales (en millions de dollars)	*IMPT*	12	Italie	I
Exportations marchandises totales (en millions de dollars)	*EXPT*	13	Japon	JAP
Calories par habitant et par jour	*CAL*	14	Norvège	N
Nombre de logements achevés pour 1 000 hab.	*LOG*	15	Pays-Bas	NL
Consommation d'électricité en kWh par habitant et par an	*ELEC*	16	Portugal	P
Dépenses publiques d'éducation en % du P.N.B.	*EDUC*	17	Grande-Bretagne	RUN
Nombre de T.V. pour 1 000 habitants	*TV*	18	Suède	S

Exemple d'interprétation.

Les trois dessins précédents visualisent les variables et les individus du problème d'analyse en composantes principales respectivement par rapport :

- aux deux premiers axes factoriels,
- aux premier et troisième axes factoriels,
- aux deuxième et troisième axes factoriels.

Les données du problème sont tirées de [BB] , chapitre VII. Ce chapitre contient également un exemple d'interprétation des dessins. En voici quelques extraits :

" ... Le premier facteur oppose le Produit National Brut, les importations, les exportations, la TV au pourcentage du P.I.B dans l'agriculture et au pourcentage d'actifs dans l'agriculture. ... On peut donc dire que le premier facteur ... peut s'interpréter comme le facteur 'revenu par tête' ... Du point de vue des pays, il oppose des pays comme la Grèce et les Etats-Unis. Le deuxième facteur oppose les recettes courantes et les dépenses d'éducation à la population totale et aux réserves officielles. ... En termes de pays, on voit s'opposer des pays scandinaves à tendance socialiste aux pays de capitalisme libéral où l'intervention de l'Etat est moins forte ... Ce facteur peut donc s'interpréter comme l'importance des dépenses publiques. ... "

Les données que l'utilisateur doit fournir à la procédure ACP sont principalement, d'une part, les projections des individus sur les axes factoriels de l'analyse dans l'espace $\mathbb{R}^p$ des variables, et d'autre part les corrélations variables-axes factoriels de l'analyse dans l'espace $\mathbb{R}^n$ des individus. Notons ici que les variables du problème doivent être centrées.

Nous allons à présent passer à l'étude de la procédure ACP. Plus précisément, nous allons aborder les points suivants :

- les possibilités offertes à l'utilisateur de la procédure,
- les contraintes imposées aux données à l'entrée de ACP,
- la justification de ces contraintes.

Les détails concernant l'utilisation de la procédure se trouvent en annexe A .

3. La procédure ACP.

La procédure est constituée de cinq parties :

- vérification des contraintes,
- tracé des deux axes graphiques,
- tracé du cercle unité,
- affichage des variables,
- affichage des individus.

Nous allons parcourir ces différents points dans l'ordre.

3.1. Les contraintes.

Le principe du traitement des erreurs est identique à celui décrit pour la procédure FCT (cf 3.2.1.§ 2.1.)

3.2. Le système d'axes graphiques.

Un axe graphique est caractérisé par sa couleur, le style de son tracé, son nom et les valeurs de ses graduations.

a) La couleur, le style et le nom.

L'étude faite à ce sujet pour la procédure FCT reste valable ici (cf 3.2.1.§ 2.2.points b,c,d).

b) Les graduations.

Ces graduations doivent indiquer l'échelle de représentation des variables et des individus :

- pour les variables, les deux axes graphiques sont gradués de -1 à +1, puisqu'il s'agit ici d'utiliser des corrélations.
- pour les individus, il y a un repère sur l'axe horizontal qui représente la longueur d'une unité de l'échelle de représentation des individus.

Nous avons donc deux valeurs possibles pour les graduations : -1 et +1. Il est donc inutile de demander à l'utilisateur de fournir ces valeurs à la procédure. Celle-ci se charge elle-même de tout de qui concerne ce problème.

3.3. Le cercle.

Ce cercle unité est tracé afin de faciliter l'interprétation du dessin. Il est tracé dans la couleur choisie par l'utilisateur. La couleur par défaut est le blanc.

3.4. Les variables.

Les positions des variables par rapport aux deux axes graphiques choisis sont fournies par les corrélations variables-axes factoriels de l'analyse dans l'espace R^n des individus. Au début de la procédure, on vérifie que ces corrélations (qui rappelons le sont fournies par l'utilisateur) sont bien comprises entre -1 et +1. Lorsqu'une erreur est détectée, un message d'erreur apparaît et l'exécution de ACP est avortée.

Sur le dessin, on représente la position d'une variable par une lettre o à côté de laquelle est inscrit le numéro de la variable. La couleur de l'ensemble des représentations peut être choisie par l'utilisateur. La couleur par défaut est le blanc.

3.5._ Les individus.

Les positions des individus par rapport aux deux axes graphiques sont fournies par la projection des individus sur les axes factoriels de l'analyse dans R^p. Ces valeurs doivent être fournies à la procédure par l'utilisateur.

Il faut remarquer ici que l'échelle utilisée pour la représentation des individus n'est pas la même que celle utilisée pour les variables. La graduation +1 des variables correspond, pour les individus, au maximum de la valeur absolue des valeurs des projections de ceux-ci sur les axes factoriels. Cela permet de représenter tous les individus à l'intérieur des limites du dessin. Cela justifie également la présence sur l'axe horizontal d'un répère dont nous avons parlé en 3.2.b, paragraphe 2.

Sur le dessin, on représente la position d'un individu par une lettre x à côté de laquelle est inscrit le numéro de l'individu. La couleur de l'ensemble des représentations peut être choisie par l'utilisateur. La couleur par défaut est le blanc.

Pour terminer, signalons que la représentation séparée (sur deux dessins différents) des variables et des individus n'est pas possible pour l'instant.

peut on projeter une partie des individus, certains centres de gravité p. ex?

3.2.4. Graphes.

Dans cette section, nous abordons le problème de représentation et de manipulation d'objets graphiques particuliers appelés graphes. Enonçons tout d'abord quelques éléments de théorie. (Voir également [FI], chapitre I).

1. Rappels théoriques

Un graphe est un schéma constitué par un ensemble de points x_1, x_2, ..., x_n, en nombre fini et que l'on appelle sommets, reliés entre-eux par des branches orientées ou non.
On dit que le graphe est :

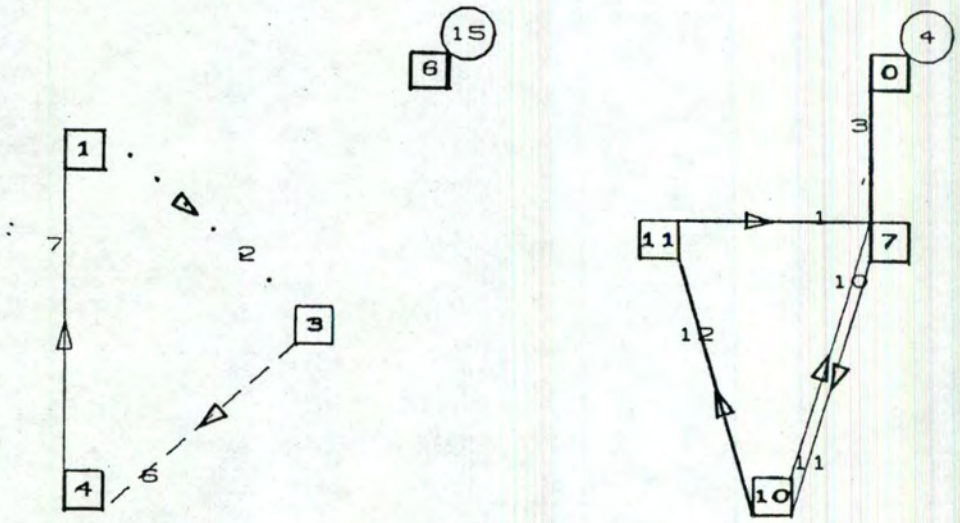
- un graphe orienté lorsque toutes les branches sont orientées, auquel cas celles-ci sont appelées les arcs du graphe;
- un graphe non orienté lorsque l'orientation des branches n'a pas d'importance, auquel cas les branches non orientées sont appelées les arêtes du graphe.

Pour définir un graphe orienté, il convient de se donner à la fois l'ensemble de ses sommets et l'ensemble de ses arcs. Le nombre n de sommets du graphe est appelé l'ordre du graphe et un arc de la forme (x_i, x_i) est appelé une boucle du graphe. Lorsqu'au plus p arcs vont d'un sommet quelconque à un autre sommet quelconque, le graphe est appelé un p-graphe.

Lorsqu'on néglige l'orientation des arcs d'un p-graphe, on parle plutôt de multigraphe. Signalons toutefois que nous ne distinguerons plus, par la suite, deux types de concepts (orientés et non orientés) associés aux graphes. En effet, du point de vue de la représentation d'un graphe, ces deux concepts peuvent être traités de façon équivalente et tout ce qui sera dit au sujet des arcs restera valable pour les arêtes.

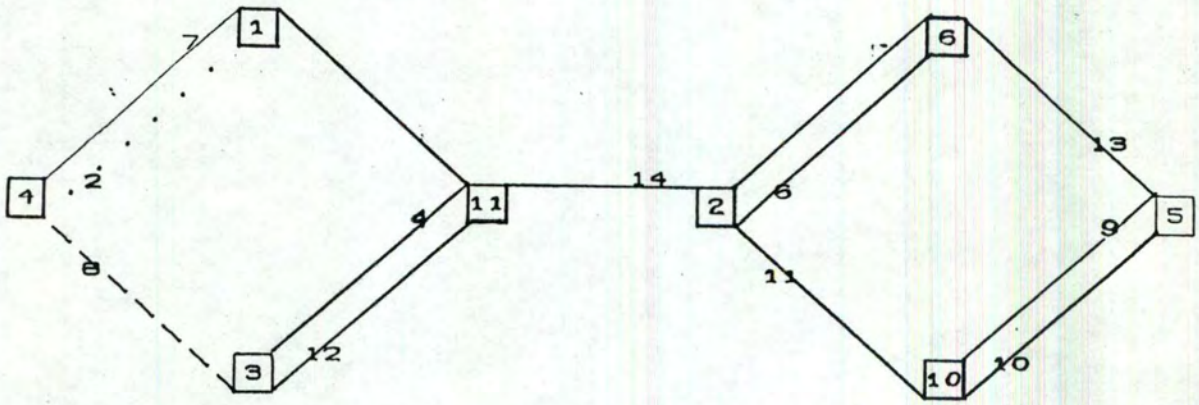
Voyons maintenant quelques exemples de réalisation.

2. Exemples de réalisation.



Le premier exemple est celui d'un graphe orienté. Cette orientation est un paramètre global à toutes les procédures d'affichage et de manipulation du graphe. Elle se traduit par une flèche (signe ▸) sur les arcs joignant deux sommets du graphe.

Le second exemple est celui d'un graphe non orienté quelconque.



Dans les deux cas, un sommet est représenté par un carré entourant le numéro de ce sommet. Un arc est représenté par un numéro qui apparaît sur le trait symbolisant cet arc. Une boucle est représentée par un petit cercle entourant son numéro, accolé au coin supérieur droit d'un sommet. Notons qu'une boucle ne porte jamais de signe d'orientation.

A partir de ces exemples, nous décrivons le schéma de la solution proposée.

3. Schéma de la solution.

Nous distinguons les deux parties suivantes :
- mémorisation du graphe dans une zone interne appelée la zone graphique;
- procédures de manipulation et de représentation des éléments du graphe.

3.1. La zone graphique

La zone graphique est une structure de données contenant, pour chaque élément de base du graphe, les informations nécessaires à l'affichage et à la manipulation de ce dernier.

Un élément de base est soit un arc, soit un sommet. Les informations sont les propriétés graphiques des éléments de base. Décrivons-les succinctement. (Voir également annexe A).

Chaque élément de base du graphe est identifié par un numéro, défini par l'utilisateur, et qui est en relation biunivoque avec l'élément en question. En outre, chaque élément possède des attributs de couleur, de style de trait, de largeur de trait et de brillance. Ces quatre dernières propriétés graphiques sont regroupées en un nombre entier de quatre chiffres appelé ci-après le code graphique. Associé à chaque élément de base, on trouve également un pointeur "détail" représenté par un entier permettant le chaînage de la zone graphique, avec d'autres zones de données éventuelles.

De plus, pour chaque élément du graphe, la zone graphique contient des informations relatives à la position de cet élément dans le plan de vision. Nous verrons ci-après de quelle façon l'utilisateur peut repérer un élément particulier parmi tous les éléments du graphe.

Signalons enfin que la zone graphique, et notamment le pointeur détail, est sous l'entière responsabilité de l'utilisateur qui y accède via les procédures de représentation et de manipulation du graphe.

3.2. Les procédures.

Les procédures de représentation et de manipulation du graphe sont conçues comme un ensemble de sous-programmes qui réalisent l'initialisation du graphe, l'ajout et la suppression d'éléments de base et la modification des propriétés graphiques de ces éléments. Elles offrent également la possibilité de déplacer une fenêtre dans le plan de vision (afin d'obtenir un effet de "ZOOM") et de produire le graphe sur la table traçante.

Ces procédures sont étudiées en détail au point 4. Voyons-en d'abord les caractéristiques générales.

A. Caractéristiques générales.

Les procédures permettent de dessiner un 1-graphe, orienté ou non, constitué de 50 sommets et de 250 arcs au maximum. Ces restrictions quant à l'ordre du graphe et le nombre d'éléments de base sont liées essentiellement à des problèmes de lisibilité du dessin. De plus, ces conditions sont suffisamment souples pour permettre de traiter bon nombre d'applications.

Le graphe est dessiné en fonction d'une grille de référence, affichée lors de l'initialisation du graphe. Cette grille est constituée de lignes et de colonnes à l'intersection desquelles sont situés les sommets.

La grille comporte 9 lignes et 11 colonnes, ce qui détermine potentiellement 99 positions. En réalité, seules sont disponibles les intersections des lignes et des colonnes d'indices pairs et les intersections des lignes et des colonnes d'indices impairs (ceci afin d'améliorer la répartition des sommets dans le plan de vision.)

L'utilisateur positionne donc les sommets du graphe en fournissant un indice-ligne et un indice-colonne valides choisis dans la grille de référence. Un arc est toujours positionné par rapport au sommet initial et au sommet terminal de cet arc.

Les procédures reçoivent les paramètres nécessaires à leur exécution et retournent une valeur entière qui est un code d'erreur (ou de réussite).

B. Conditions d'erreurs.

On peut distinguer deux types d'erreurs :

- les erreurs susceptibles de rattrapage (par exemple, un code graphique invalide), pour lesquelles la procédure se poursuit normalement en utilisant des valeurs par défaut pour les paramètres erronés.
- les erreurs non susceptibles de rattrapage. Dans ce cas, l'exécution de la procédure est sans effet. Notons qu'aucun message d'erreur n'est envoyé à l'utilisateur. Celui-ci doit tester le code d'erreur pour déterminer les paramètres erronés.

Cette dernière solution a été choisie afin d'empêcher la surimpression des messages d'erreurs et du dessin, sur le terminal graphique. Une procédure particulière permet le dialogue par l'intermédiaire d'un autre terminal.

4. Description des procédures.

Nous allons décrire les différentes procédures en donnant pour chacune d'elles :

- la fonction qu'elle réalise
- les contraintes quant à son utilisation,
- les conditions d'erreurs (éventuelles).

Les spécifications complètes et le manuel d'utilisation se trouvent en annexe A .

4.1. Ajout d'un arc ou d'un sommet.

La procédure ADDSOM permet d'ajouter, dans le plan de vision, un sommet dont on fournit les propriétés graphiques.

Le numéro du sommet est un entier compris entre 0 et 99.

Huit couleurs sont disponibles, ainsi que 4 styles de trait, 3 largeurs de trait et 3 indices de brillance. Par défaut, la couleur du sommet est une des 8 couleurs (différente de la couleur du fond), son style est le trait continu, la largeur de trait et l'indice de brillance ont une valeur moyenne.

Les indices ligne et colonne sont des nombres entiers à choisir dans la grille de référence.

Les conditions d'erreurs sont:

- la zone graphique est remplie,
- le sommet existe déjà dans le graphe,
- le numéro ou la position du sommet sont erronés.

La procédure ADDARC permet d'ajouter, dans le plan de vision, un arc dont on fournit les propriétés graphiques.

Le numéro de l'arc est un entier compris entre 0 et 999.

Les attributs graphiques sont les mêmes que ceux d'un sommet.

L'utilisateur doit fournir le numéro des sommets initial et terminal.

Les conditions d'erreurs sont :
- la zone graphique est remplie,
- l'arc existe déjà dans le graphe,
- le numéro de l'arc ou celui du sommet initial ou terminal sont erronés.

4.2. Suppression d'un arc ou d'un sommet.

Les procédures DELSOM et DELARC permettent respectivement de supprimer, du plan de vision, un sommet et un arc dont on fournit le numéro.

Un code d'erreur est renvoyé si le sommet, ou l'arc, n'existe pas.

4.3. Modification des propriétés graphiques.

Les procédures MDNSOM et MDNARC permettent de modifier, dans le plan de vision, le numéro d'un sommet et d'un arc respectivement.

Un code d'erreur est renvoyé si le sommet, ou l'arc, à modifier n'existe pas ou si le nouveau numéro est erroné.

Les procédures MDCSOM et MDCARC permettent de modifier, dans le plan de vision, le code graphique d'un sommet et d'un arc respectivement.

Un code d'erreur est renvoyé si le sommet, ou l'arc, n'existe pas.

Les procédures MDPSOM et MDPARC permettent de modifier, dans le plan de vision, la position d'un sommet et d'un arc respectivement.

Un code d'erreur est renvoyé si le sommet, ou l'arc, n'existe pas, ou si la nouvelle position est erronée.

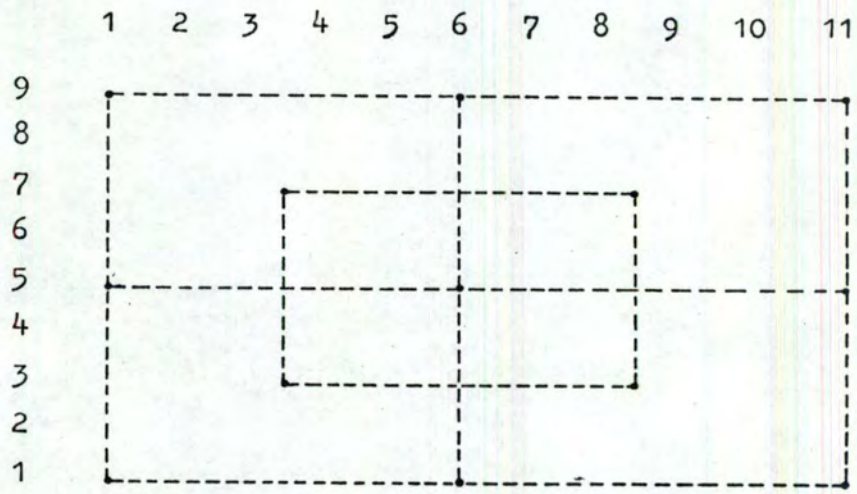
Afin d'obtenir un affichage correct, chacune de ces procédures s'exécute en deux phases :

- effaçage de l'arc ou du sommet dont on veut modifier des propriétés graphiques,
- retracé de l'arc ou du sommet avec ses nouvelles propriétés.

4.4. Déplacement de la fenêtre.

La procédure MOVEGR permet de déplacer une fenêtre dans le plan de vision.

Ce dernier est découpé en 5 zones, chaque zone correspond à une fenêtre. Le schéma suivant illustre cette découpe.



Le choix d'une fenêtre particulière provoque un effet de "ZOOM" sur la partie du graphe comprise dans la fenêtre.

Un code d'erreur est renvcyé si le choix de la fenêtre est erroné.

4.5. Modification de la couleur du fond.

La procédure PBACKG permet de modifier la couleur de fond du terminal graphique.

Cette possibilité peut être utilisée pour "masquer" temporairement des arcs ou des sommets du graphe qui seraient colorés dans la couleur de fond.

Huit couleurs sont disponibles. Un code d'erreur est renvoyé si la couleur de fond choisie est erronée.

4.6. Validation de l'image.

Cette fonction s'appuie sur le principe de segmentation des images introduit au chapitre II, paragraphe 2.D.4.

En effet, chaque élément de base (arc ou sommet) correspond, lors de sa création, à un segment temporaire distinct. L'ensemble du graphe correspond, quant à lui, à un seul segment retenu.

De ceci, il découle deux problèmes :

1.- Il est virtuellement impossible de déplacer une fenêtre ou de modifier la couleur de fond. Pour des raisons inhérentes au logiciel DI-3000, ces deux fonctions provoquent des changements d'image implicites et entraînent donc l'effacement des segments temporaires.

2.- Il est virtuellement impossible d'ajouter des éléments au segment retenu que constitue le graphe. On sait, en effet, qu'un segment (temporaire ou retenu) ne peut plus être modifié après sa création.

Ces deux problèmes sont résolus à l'aide de la validation de l'image qui consiste à détruire le segment retenu courant et à le recréer à partir des informations de la zone graphique, incluant ainsi les segments temporaires couramment ajoutés par l'utilisateur.

Cette validation est réalisée par la procédure PVALID.

Elle ne renvoie aucun code d'erreur et doit être appelée avant tout déplacement de la fenêtre ou toute modification de la couleur de fond.

4.7. Chargement initial et sauvetage du graphe.

La procédure GLOAD permet d'afficher en une fois un graphe dont on fournit les propriétés graphiques sous la forme d'un tableau de données.

L'intérêt d'une telle procédure est que l'utilisateur est dispensé du positionnement explicite de chaque élément du graphe. En effet, un algorithme particulier place automatiquement les sommets par rapport à la grille de référence. A partir de ce graphe initial, l'utilisateur peut procéder aux différentes manipulations possibles.

Les conditions d'erreurs sont :

- la capacité de la zone graphique a été dépassée lors du chargement,
- le numéro d'au moins un élément du graphe est erroné.

La procédure GSAVE permet le sauvetage du graphe dans une zone de données, pour un rechargement ultérieur.

Cette procédure renvoie un code d'erreur si on tente de sauver un nombre d'éléments supérieur au nombre maximum autorisé.

4.8. Création d'un métafichier.

La procédure PPRINT permet de créer un métafichier contenant l'image du graphe couramment construit.

L'utilisateur fournit le nom du métafichier. Celui-ci est limité à 10 caractères au maximum. Par défaut, ce nom est "DI-3000.MTA".

Le métafichier peut ensuite être utilisé pour produire le graphe sur la table traçante.

Cette procédure ne renvoie aucun code d'erreur.

4.9. Lecture et écriture du pointeur "détail".

Les procédures GRSPNR et GRAPNR permettent de lire le pointeur détail associé à un sommet et à un arc respectivement.

Les procédures GWSPNR et GWAPNR permettent de modifier le pointeur détail associé à un sommet et à un arc respectivement.

Ces procédures renvoient un code d'erreur si le sommet, ou l'arc, référencé n'existe pas.

Nous terminons ici l'étude consacrée à la représentation et à la manipulation des graphes. Signalons encore qu'un exemple de création de graphe est traité en détails dans les annexes .

Nous abordons maintenant le problème de représentation d'une classification hiérarchique.

3.2.5. Classifications hiérarchiques.

Dans cette section, nous abordons la représentation d'arbres de classification hiérarchique (encore appelés "dendrogrammes").

Voyons d'abord quelques éléments de théorie tirés de [ββ] .

1. Rappels théoriques.

Les méthodes de classification hiérarchique s'appliquent à des populations E d'individus. Elles nécessitent les définitions d'une distance entre individus prix deux à deux, puis d'une distance entre groupes d'individus.

Elles procèdent itérativement en déterminant, à chaque itération, les deux groupes les plus proches (au sens de la distance choisie) et en effectuant leur réunion. Au départ, les groupes sont constitués d'un seul individu. Le processus s'arrête lorsqu'on est arrivé à un groupe rassemblant tous les individus. On a ainsi constitué un arbre de classification, le dendrogramme.

Notons encore que la distance séparant deux groupes d'individus est en fait une distance ultramétrique (ou ultramétique) et est calculée à partir des distances entre les individus. La méthode de Johnson [ββ] constitue un exemple de construction d'une distance ultramétrique à partir d'une distance sur E x E, et par suite, d'un arbre de classification.

Voyons maintenant un exemple de réalisation.

2. Exemple de réalisation.

	1	2	3	4	5
1	0	0.8	0.8	0.8	0.5
2		0	0.2	0.4	0.8
3			0	0.4	0.8
4				0	0.8
5					0

TABLEAU I

Exemple d'ultramétrique .

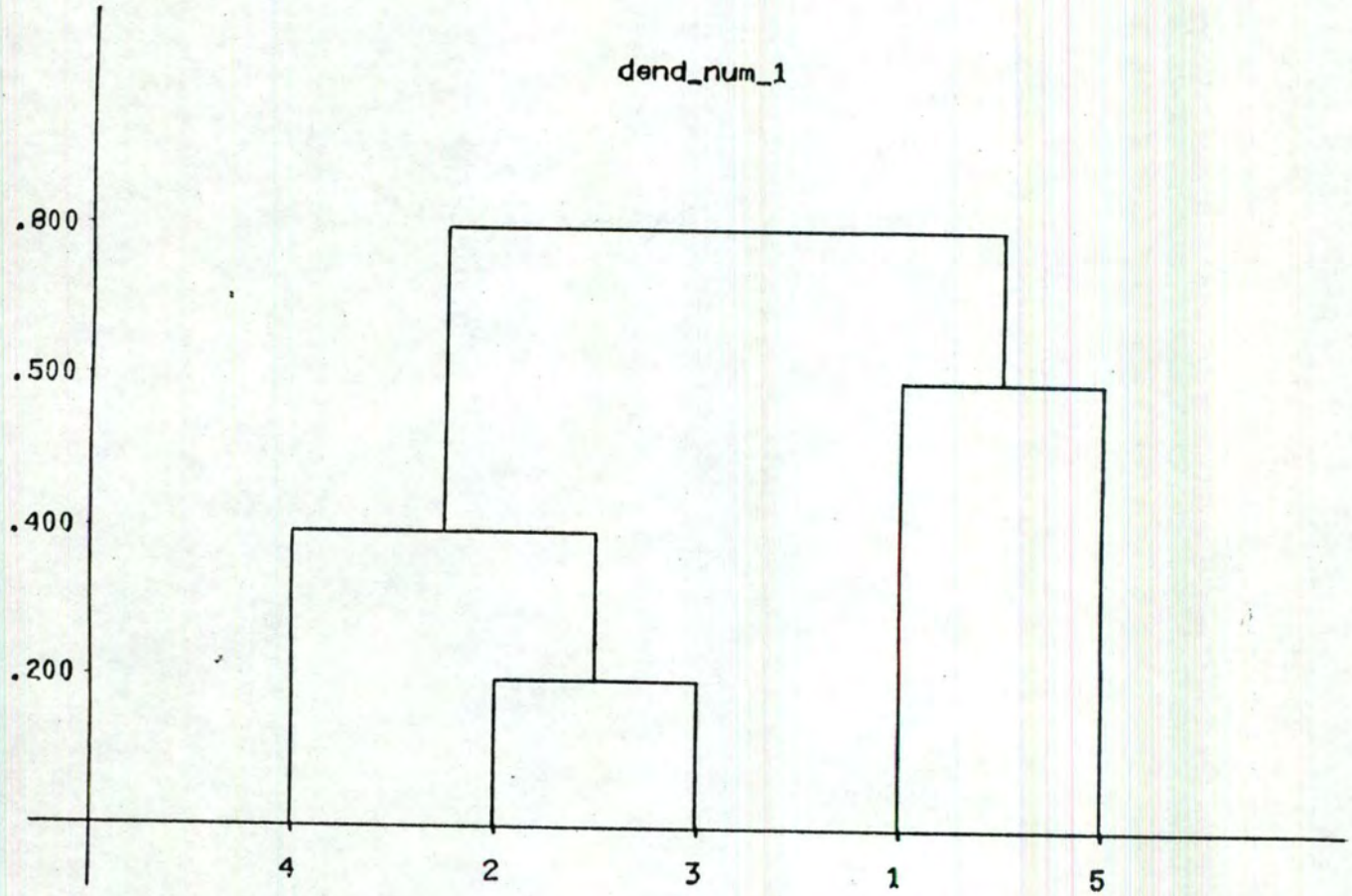


FIGURE 1

Dendrogramme correspondant à l'ultramétrique du Tableau I

A partir du tableau de l'ultramétrique, on construit le dendrogramme représenté par un arbre binaire. Le numéro des individus apparaît en abscisses. En ordonnées, on trouve l'indice d'agrégation des différents groupes.

Le nom du dessin apparaît en haut, au centre.

A l'aide de cet exemple, nous décrivons le schéma de la solution proposée.

3. Schéma de la solution.

La représentation du dendrogramme est obtenue à l'aide de la procédure ULTRAM. D'autres procédures sont fournies pour construire l'ultramétrique et produire le dessin sur la table traçante.

Ces procédures sont étudiées en détails au point 4. Voyons-en les caractéristiques générales.

3.1. Caractéristiques générales.

Les procédures permettent de dessiner un dendrogramme correspondant à une population de 40 individus au maximum. Cette restriction est liée essentiellement à des problèmes de lisibilité du dessin. Elle permet toutefois de traiter bon nombre d'applications.

Notons également qu'au lieu de définir un indice de distance, entre les individus, on aurait pu définir un indice de similarité. Dans ce dernier cas, on groupe les individus tels que leur similarité est maximale. Les procédures offrent la possibilité de choisir entre ces deux options.

Les procédures reçoivent les paramètres nécessaires à leur exécution et renvoient une valeur entière qui est un code d'erreur (ou de réussite).

3.2. Conditions d'erreurs.

Le principe du traitement des erreurs est identique à celui des procédures de représentation d'un graphe (cf. 3.2.4., paragraphe 3.2., b)

4. Description des procédures.

Nous décrivons ici les différentes procédures, en donnant pour chacune d'elles :

- la fonction qu'elle réalise,
- les contraintes quant à son utilisation,
- les conditions d'erreurs (éventuelles).

Les spécifications complètes et le manuel d'utilisation se trouvent en annexe A .

4.1. Tracé du dendrogramme.

La procédure ULTRAM permet de tracer un dendrogramme à partir du tableau de l'ultramétrique fourni par l'utilisateur.

Le nom du dessin est une chaîne de 10 caractères au maximum.

L'arbre de classification est caractérisé par un code graphique reprenant les attributs de couleur, de style, de largeur de trait et de brillance. Huit couleurs sont disponibles (la huitième, le noir, est toujours la couleur de fond) ainsi que 4 styles, 3 largeurs de trait et 3 indices de brillance.

Par défaut, la couleur de l'arbre est le rouge, son style est le trait continu, la largeur de trait et l'indice de brillance ont une valeur moyenne.

Les axes sont également caractérisés par un code graphique dont la valeur par défaut est la même que celle de l'arbre. Les graduations en abscisses sont les numéros des individus attribués automatiquement dans l'ordre où les individus apparaissent dans le tableau de l'ultramétrique. Chaque graduation en ordonnées, tirée directement du tableau de l'ultramétrique, est un nombre réel positif de 5 caractères, y compris le point décimal.

La procédure ULTRAM renvoie un code d'erreur si le nombre d'individus est inférieur à 2 ou supérieur à 40.

4.2. Création d'un métafichier.

La procédure DPRINT permet de créer un métafichier contenant l'image du dendrogramme couramment construit.

L'utilisateur fournit le nom du métafichier. Celui-ci est limité à 10 caractères au maximum. Par défaut, ce nom est "DI-3000.MTA".

Le métafichier permet de produire le dendrogramme sur la table traçante.

Cette procédure ne renvoie aucun code d'erreur.

4.3. Construction de l'ultramétrique.

Bien que sortant du cadre de cette étude, il nous a semblé intéressant de fournir à l'utilisateur un outil complet pour la représentation des dendrogrammes.

Ainsi, la procédure JOHNSO permet de construire une ultramétrique suivant la méthode de Johnson.

L'utilisateur a le choix entre les formules du minimum, du maximum ou de la moyenne pour calculer les distances entre les groupes à partir des distances entre individus. Par défaut, la formule utilisée est celle de la moyenne.

Cette procédure peut donc être appelée avant l'appel à la procédure ULTRAM. Elle ne renvoie aucun code d'erreur.

C H A P I T R E 4

EVALUATION ET PERSPECTIVES.

Dans ce dernier chapitre, nous aborderons, dans un premier temps, les limitations de notre travail qui sont dues aux outils de base utilisés. Ensuite, nous verrons en quelques lignes quelles sont les perspectives existantes, dans le domaine graphique à l'Institut.

4.1. Evaluation des outils de base.

En ce qui concerne le matériel graphique, et plus précisément le terminal GIGI (confer chapitre 2, paragraphe 2.1.), il existe deux phénomènes pouvant nuire à la qualité des dessins : la contamination des couleurs et l'approximation de certains tracés.

L'écran vidéo du GIGI est divisé en bandes horizontales et verticales formant un damier de ± 64 x 64. Lorsque deux objets de couleurs différentes sont tracés assez près l'un de l'autre, a fortiori donc lorsque deux lignes se coupent, il se peut qu'il y ait contamination. Cela veut dire que la couleur du deuxième objet peut recouvrir celle du premier jusqu'à la limite de la bande horizontale ou verticale la plus proche.

L'approximation des tracés est due, quant à elle, à la résolution de l'écran. Cette approximation peut devenir sensible lorsqu'on trace une ligne presque horizontale ou verticale. Dans ce cas, la droite prend l'apparence d'une ligne brisée.

Notons que ces deux phénomènes ne se produisent pas lors de l'utilisation de la table traçante. Pour cette dernière, rappelons toutefois que, jusqu'à présent, seules quatre couleurs sur les huit possibles sont mises à la disposition des utilisateurs.

En ce qui concerne les outils logiciels de base, nous ne parlerons ici que du générateur Pascal de métafichier et plus précisément du problème des qualités de texte disponibles.

Les utilisateurs de DI.3000 disposent de quatre qualités de texte allant de la plus mauvaise à la meilleure, alors que ceux du générateur Pascal de métafichier ne disposent que d'une seule qualité, la plus mauvaise.

Le principal inconvénient de ce fait est qu'avec ce jeu de caractères, appelés caractères matériels, il est impossible d'écrire autrement qu'horizontalement et de gauche à droite.

4.2. Perspectives.

La plupart des sujets traités dans ce travail ne nécessitaient pas l'implantation de routines de création et de manipulation interactive d'objets graphiques. Par routines interactives, nous entendons ici des routines dont l'effet peut être visualisé immédiatement sur un écran. (De plus, il faut signaler que ce caractère interactif ne semble pas compatible avec les objectifs de la partie Pascal de notre travail. (Cf. 3.1.))

Néanmoins, dans d'autres domaines, tels que la représentation de modèles du type "entité-association", ou la conception de schémas électriques ou de plans d'ingéniérie, de telles routines constituent un outil beaucoup plus adapté. En fait, nous touchons ici au domaine de la Conception Assistée par Ordinateur.

En conclusion, nous constatons qu'il reste encore beaucoup de possibilités pour des travaux ultérieurs dans le domaine graphique. Nous espérons néanmoins avoir contribué par ce travail à une meilleure compréhension du sujet.

BIBLIOGRAPHIE.

[BB] P. Bertier, JM. Bouroche " Analyse de données multidimensionelles " PUF, 1979.

[BBP] R. Bergeron, P. Bono, J. Foley " Graphics Programming Using the Core System " Computing Survey 10,4 (Dec 78) pages 365 à 380.

[CP] I. Carlbom, J. Paciorek " Planar Geometric Projections and Viewing Transformations " (id.) pages 465 à 502.

[DI] DI-3000 User's Guide Precision Visuals Inc. janvier 1982.

[FI] J. Fichefet " Théorie des graphes et applications " notes de cours, FNDP, 1980.

[MVD] J. Michener, A. Van Dam " A functional overview of the core system with glossary " Computing Survey 10,4 (Dec 78) pages 381 à 387.

[NVD] W. Newman, A. Van Dam " Recent Efforts Toward Graphics Standardization " (id.) pages 365 à 380.

FACULTES UNIVERSITAIRES NOTRE-DAME DE LA PAIX (NAMUR)

INSTITUT D'INFORMATIQUE

REALISATION D'UN PROGICIEL GRAPHIQUE

ANNEXE B

LES PROCEDURES

Ph. Mataigne

Y. Perozzo

Année académique 1981 - 1982

FACULTES UNIVERSITAIRES NOTRE-DAME DE LA PAIX (NAMUR)

INSTITUT D'INFORMATIQUE

REALISATION D'UN PROGICIEL GRAPHIQUE

ANNEXE B

LES PROCEDURES

Ph. Mataigne

Y. Perozzo

Année académique 1981 - 1982

ANNEXE 2 : LA LIBRAIRIE.

PARTIE PASCAL.

1 ERE PARTIE : LE GENERATEUR DE METAFICHIER.

```
(*$M-,d-,c-*)
program xlibd;

type tmp = 0..3;

     com = packed record
           coml : 0..1b;
           class : 0..7b;
           index : 0..17b;
           parnbr : 0..377b;
           comlb : 0..1b;
           classb : 0..7b;
           indexb : 0..17b;
           parnbb : 0..377b;
           filcom : 0..17b;
           end;


     car = packed record
           p1 : 0..1b;
           ch1 : 0..177b;
           p2 : 0..1b;
           ch2 : 0..177b;
           p1b : 0..1b;
           ch1b : 0..177b;
           p2b : 0..1b;
           ch2b : 0..177b;
           filcar : 0..17b;
           end;

     movdrw = packed record
                 b15 : 0..1b;
                 b014 : 0..77777b;
                 b15b : 0..1b;
                 b014b : 0..77777b;
                 filmdr:0..17b;
        end;

     param = packed record
             par1: 0..177777b;
             par2: 0..177777b;
             filpar : 0..17b;
        end;

     mot = packed record
```

```
        case i: tmp of
         0: (val1 : movdrw);
         1: (val2 : com);
         2: (val3 : param);
         3: (val4 : car);
         4: (val5 : integer);
   end.


rtab1 = array [1..500] of real;
rtab2 = array [1..50] of real;
ndct1 = array [1..502] of integer;
ndct2 = array [1..50] of integer;
string10 = packed array [1..10] of char;
achar = array [1..50] of string10;
arw = record
          arwx : rtab2;
          arwy : rtab2;
      end;
andc = record
           andcx : ndct2;
           andcy : ndct2;
       end;
frw = record
           frwx : rtab1;
           frwy : rtab1;
      end;
fndc = record
           fndcx : ndct1;
           fndcy : ndct1;
       end;
zone = packed record
                  col : 0..777b;
                  lst : 0..777b;
                  itype : 0..77b;
                  fc : 0..77b;
                  ist : 0..77b;
                  dim : integer;
                  nom : string10;
       end;
zones = array[1..40] of zone;
opt = record
          cars : integer;
          acol : integer;
          ast : integer;
          axtyp : integer;
          adrw : boolean;
          axnb : integer;
          aynb : integer;
          axname : string10;
          ayname : string10;
          fxconv : boolean;
          fyconv : boolean;
          fpronb : integer;
          fnb : integer;
          znb : integer;
          z : zones;
end;
gval = record
           grsize : integer;
```

```
                notl : integer;
                not2 : integer;
                not3 : integer;
                not4 : integer;
                anam : integer;
                dmax : integer;
                nbpt : integer
    end;

    par = record
       a : andc;
       c : fndc;
       tca : achar;
       tc : achar;
       px : integer;
       py : integer;
       gv : gval
    end;

    metafile=file of integer;
    mfdesc=record
                word:mot;
                nbre:integer;
                pos:boolean
           end;

var ival:integer;
    b,rinv:boolean;
    fd:fndc;
    p:par;
    f:mfdesc;

(************************************************)

procedure inmeta(var mf:metafile);
(*-----------------------------*)

begin
    f.word.val1.filmdr:=0;
    f.nbre:=f.nbre+1;
    mf^:=f.word.val5;put(mf);
end;



procedure md0(var mf:metafile;x,y,c10:integer);
(*---------------------------------------*)

begin
    if f.pos then begin
                with f.word.val1 do
                    begin
                        b15:=0;
                        b014:=x;
                        b15b:=c10;
                        b014b:=y;
                    end;
                inmeta(mf);
                end
    else begin
```

```
                with f.word.val1 do
                    begin
                        b15b:=0;
                        b014b:=x;
                    end;
                inmeta(mf);
                with f.word.val1 do
                    begin
                        b15:=c10;
                        b014:=y;
                    end;
            end;
end;


procedure parnb0(var mf:metafile;c10,ind0:integer);
(*-----------------------------------------------*)

begin
    if f.pos then begin
                    with f.word.val2 do
                        begin
                            com1:=1;
                            class:=c10;
                            index:=ind0;
                            parnbr:=0;
                        end;
                    f.pos:=false;
                end
    else begin
                    with f.word.val2 do
                        begin
                            com1b:=1;
                            classb:=c10;
                            indexb:=ind0;
                            parnbb:=0;
                        end;
                    inmeta(mf);
                    f.pos:=true;
     end;
end;


procedure parnb1(var mf:metafile;c10,ind0,i:integer);
(*-----------------------------------------------*)

begin
    if f.pos then begin
                with f.word.val2 do
                    begin
                        com1:=1;
                        class:=c10;
                        index:=ind0;
                        parnbr:=1;
                    end;
                f.word.val3.par2:=i;
                inmeta(mf);
                end
```

```
    else begin
              with f.word.val2 do
                     begin
                         comlb:=1;
                         classb:=cl0;
                         indexb:=ind0;
                         parnbb:=1;
                     end;
              inmeta(mf);
              f.word.val3.par1:=i;
         end;
end;



procedure parnb2(var mf:metafile;cl0,ind0,w,h:integer);
(*-------------------------------------------------*)

begin
if f.pos then
    begin
        with f.word.val2 do
            begin
                com1:=1;
                class:=cl0;
                index:=ind0;
                parnbr:=2;
            end;
        f.word.val3.par2:=w;
        inmeta(mf);
        f.word.val3.par1:=h;
        f.pos:=false;
    end
else begin
         with f.word.val2 do
             begin
                comlb:=1;
                classb:=cl0;
                indexb:=ind0;
                parnbb:=2;
            end;
         inmeta(mf);
         f.word.val3.par1:=w;
         f.word.val3.par2:=h;
         inmeta(mf);
         f.pos:=true;
     end;
end;



procedure parnb3(var mf:metafile;cl0,ind0,xcomp,ycomp,zcomp:
                 integer);
(*-----------------------------------------------------*)

begin
if f.pos then
    begin
        with f.word.val2 do
```

```
            begin
                com1:=1;
                class:=cl0;
                index:=ind0;
                parnbr:=3;
            end;
        f.word.val3.par2:=xcomp;
        inmeta(mf);
        f.word.val3.par1:=ycomp;
        f.word.val3.par2:=zcomp;
        inmeta(mf);
    end
else begin
         with f.word.val2 do
             begin
                com1b:=1;
                class:=cl0;
                indexb:=ind0;
                parnbb:=3;
            end;
         inmeta(mf);
         f.word.val3.par1:=xcomp;
         f.word.val3.par2:=ycomp;
         inmeta(mf);
         f.word.val3.par1:=zcomp;
     end;
end;



procedure noop(var mf:metafile);
(*--------------------------*)

var cl,ind:integer;

begin
cl:=0;
ind:=4;
parnb0(mf,cl,ind);
end;



procedure rempl(var mf:metafile);
(*--------------------------*)

var i,k:integer;

begin
if f.pos then begin
                k:=360-(f.nbre*2);
(* k est tjrs pair *)
            end
       else begin
                k:=360-((f.nbre*2)+1);
(* k est tjrs impair *)
            end;
for i:=1 to k do
noop(mf);
```

```
f.nbre:=0;
end;


procedure verif(var mf:metafile;pp:integer);
(*--------------------------------------*)

var tmp,tmp1,tmp2,k,nb:integer;

begin
if f.pos then tmp :=(f.nbre*2)+pp
       else tmp :=((f.nbre*2)+pp)+1;
    k:=360-tmp;
    if k<0 then begin
                    nb:=abs(k);
(* nb = le depassement ds le buffer suivant *)
                    if nb=pp then f.nbre:=0
(* ds ce cas pos ne peut etre que true      *)
                             else begin
                                      tmp2:=pp-nb;
                                      for tmp1:=1 to tmp2 do
                                          noop(mf);
                                      f.nbre:=0;
                                  end;
                end;
end;


procedure amove(var mf:metafile;x,y:integer);
(*--------------------------------------*)

var tmp:integer;

begin
verif(mf,2);
tmp:=0;
md0(mf,x,y,tmp);
end;


procedure adraw(var mf:metafile;x,y:integer);
(*--------------------------------------*)

var tmp:integer;

begin
verif(mf,2);
tmp:=1;
md0(mf,x,y,tmp);
end;


procedure bmeta(var mf:metafile);
(*----------------------------*)

begin
    with f.word.val2 do
```

```
        begin
            com1:=1;
            class:=0;
            index:=0;
            parnbr:=1;
        end;
    f.word.val3.par2:=1;
    inmeta(mf);
end;


procedure emeta(var mf:metafile);
(*--------------------------*)

var c1,ind:integer;

begin
verif(mf,1);
c1:=0;ind:=1;
parnb0(mf,c1,ind);
end;


procedure defvw(num:integer;var mf:metafile);
(*-------------------------------------*)

begin
    with f.word.val2 do
        begin
            com1:=1;
            class:=0;
            index:=2;
            parnbr:=3;
        end;
    f.word.val3.par2:=32767;
    inmeta(mf);
    f.word.val3.par1:=num;
    f.word.val3.par2:=32767;
    inmeta(mf);
end;


procedure bpict(var mf:metafile;i:integer);
(*-----------------------------------*)

label 1;

begin
if ival=0 then begin
    writeln(tty,'l''appel a INITMF n''a pas ete fait.');
    b:=true;
    goto 1
                end;
ival:=2;
rempl(mf);
with f.word.val2 do
    begin
        com1:=1;
        class:=1;
        index:=0;
```

```
        parnbr:=1;
    end;
if (i<=0) or (i>32767) then i:=1;
f.word.val3.par2:=i;
inmeta(mf);
1:end;



procedure epict(var mf:metafile);
(*---------------------------*)

label 1;

var c1,ind:integer;

begin
if b then goto 1;
if ival<>2 then begin
    writeln(tty,'un appel a INITMF ou/et a BPICT n''a ');
    writeln(tty,'pas ete fait.');
    b:=true;
    goto 1
                end;
ival:=3;
verif(mf,1);
c1:=1;ind:=1;
parnb0(mf,c1,ind);
1:end;



procedure set2d(var mf:metafile);
(*---------------------------*)

begin
    with f.word.val2 do
        begin
            com1:=1;
            class:=2;
            index:=0;
            parnbr:=0;
        end;
    f.pos:=false;
end;



procedure setms(var mf:metafile;ms:integer);
(*-------------------------------------*)

var c1,ind:integer;

begin
verif(mf,2);
c1:=2;ind:=2;
parnb1(mf,c1,ind,ms);
end;
```

```
procedure outsm(var mf:metafile;x,y:integer);
(*-------------------------------------*)

var cl,ind:integer;

begin
verif(mf,3);
cl:=2;ind:=3;
parnb2(mf,cl,ind,x,y);
end;



procedure bpolyl(var mf:metafile);
(*-----------------------------*)

var cl,ind:integer;

begin
verif(mf,1);
cl:=2;ind:=5;
parnb0(mf,cl,ind);
end;



procedure epolyl(var mf:metafile);
(*-----------------------------*)

var cl,ind:integer;

begin
verif(mf,1);
cl:=2;ind:=6;
parnb0(mf,cl,ind);
end;



procedure bpolym(var mf:metafile);
(*-----------------------------*)

var cl,ind:integer;

begin
verif(mf,1);
cl:=2;ind:=7;
parnb0(mf,cl,ind);
end;



procedure epolym(var mf:metafile);
(*-----------------------------*)
```

```
var cl,ind:integer;

begin
verif(mf,1);
cl:=2;ind:=8;
parnb0(mf,cl,ind);
end;



procedure bpolyg(var mf:metafile);
(*----------------------------*)

var cl,ind:integer;

begin
verif(mf,1);
cl:=2;ind:=9;
parnb0(mf,cl,ind);
end;



procedure epolyg(var mf:metafile);
(*----------------------------*)

var cl,ind:integer;

begin
verif(mf,1);
cl:=2;ind:=10;
parnb0(mf,cl,ind);
end;



procedure outstr(var mf:metafile;nom:string10;lgr:integer);
(*-----------------------------------------------------*)

label 01,101;
var tmp1,tmp2,tmp3,i:integer;

begin
tmp1:=lgr;
(* nbre de car dans le string *)
tmp2:=(tmp1+1)div 2 ;
(* nbre de chunks occupes par des car *)
tmp3:=tmp2+2;
(* nbre de chunks total *)
verif(mf,tmp3);
if f.pos then
    begin
        with f.word.val2 do
            begin
                com1:=1;
                class:=3;
```

```
                index:=0;
                parnbr:=tmp3-1;
            end;
        f.word.val3.par2:=tmp1;
        inmeta(mf);
        i:=1;
(* indice de la lettre qu'on va ecrire *)
    01:if tmp2>=2 then
(* il y a au moins 3 caracteres *)
        begin
        f.word.val4.p1:=0;
        f.word.val4.ch1:=ord(nom[i]);
        f.word.val4.p2:=0;
        f.word.val4.ch2:=ord(nom[i+1]);
        f.word.val4.p1b:=0;
        f.word.val4.ch1b:=ord(nom[i+2]);
        f.word.val4.p2b:=0;
        f.word.val4.ch2b:=ord(nom[i+3]);
        inmeta(mf);
        tmp2:=tmp2-2;i:=i+4;
        goto 01;
        end;
if tmp2 = 1 then
(* il y a au moins 2 caracteres *)
        begin
        f.word.val4.p1:=0;
        f.word.val4.ch1:=ord(nom[i]);
        f.word.val4.p2:=0;
        f.word.val4.ch2:=ord(nom[i+1]);
        f.pos:=false;
        end
end
else begin
        with f.word.val2 do
            begin
                com1b:=1;
                class:=3;
                indexb:=0;
                parnbb:=tmp3-1;
            end;
        inmeta(mf);
        f.word.val3.par1:=tmp1;
        i:=1;
(* indice de la lettre qu'on va ecrire *)
    101:if tmp2 >=2 then
        begin
        f.word.val4.p1b:=0;
        f.word.val4.ch1b:=ord(nom[i]);
        f.word.val4.p2b:=0;
        f.word.val4.ch2b:=ord(nom[i+1]);
        inmeta(mf);
        f.word.val4.p1:=0;
        f.word.val4.ch1:=ord(nom[i+2]);
        f.word.val4.p2:=0;
        f.word.val4.ch2:=ord(nom[i+3]);
        tmp2:=tmp2-2;i:=i+4;
        goto 101;
        end;
if tmp2 = 1 then
        begin
```

```
        f.word.val4.p1b:=0;
        f.word.val4.ch1b:=ord(nom[i]);
        f.word.val4.p2b:=0;
        f.word.val4.ch2b:=ord(nom[i+1]);
        inmeta(mf);
        f.pos:=true;
        end;
    end;
end;


procedure settft(var mf:metafile;font:integer);
(*-----------------------------------------*)

var cl,ind:integer;

begin
verif(mf,2);
cl:=3;ind:=1;
parnb1(mf,cl,ind,font);
end;


procedure setcsz(var mf:metafile;w,h:integer);
(*---------------------------------------*)

var cl,ind:integer;

begin
verif(mf,3);
cl:=3;ind:=2;
parnb2(mf,cl,ind,w,h);
end;


procedure setcg(var mf:metafile;ratioh,ratiov:integer);
(*-------------------------------------------------*)


var cl,ind:integer;

begin
verif(mf,3);
cl:=3;ind:=6;
parnb2(mf,cl,ind,ratioh,ratiov);
end;



procedure settp(var mf:metafile;tp:integer);
(*-------------------------------------*)

var cl,ind:integer;

begin
verif(mf,2);
cl:=3;ind:=8;
parnb1(mf,cl,ind,tp);
end;
```

```
procedure settj(var mf:metafile;hj,vj:integer);
(*-------------------------------------*)

var cl,ind:integer;

begin
verif(mf,3);
cl:=3;ind:=7;
parnb2(mf,cl,ind,hj,vj);
end;



procedure setcbs(var mf:metafile;xcomp,ycomp,zcomp:integer);
(*-----------------------------------------------------*)

var cl,ind:integer;

begin
verif(mf,4);
cl:=3;ind:=4;
parnb3(mf,cl,ind,xcomp,ycomp,zcomp);
end;


procedure setcol(var mf:metafile;i:integer);
(*-------------------------------------*)

var cl,ind:integer;

begin
verif(mf,2);
cl:=4;ind:=1;
parnb1(mf,cl,ind,i);
end;



procedure setst(var mf:metafile;i:integer);
(*-------------------------------------*)

var cl,ind:integer;

begin
verif(mf,2);
cl:=4;ind:=3;
parnb1(mf,cl,ind,i);
end;


procedure setlw(var mf:metafile;i:integer);
(*-------------------------------------*)

var cl,ind:integer;

begin
verif(mf,2);
```

```
cl:=4;ind:=4;
parnbl(mf,cl,ind,i);
end;



procedure setpen(var mf:metafile;i:integer);
(*--------------------------------------*)

var cl,ind:integer;

begin
verif(mf,2);
cl:=4;ind:=5;
parnbl(mf,cl,ind,i);
end;



procedure polest(var mf:metafile;i:integer);
(*--------------------------------------*)

var cl,ind:integer;

begin
verif(mf,2);
cl:=4;ind:=6;
parnbl(mf,cl,ind,i);
end;



procedure polist(var mf:metafile;i:integer);
(*--------------------------------------*)

var cl,ind:integer;

begin
verif(mf,2);
cl:=4;ind:=7;
parnbl(mf,cl,ind,i);
end;



procedure polfc(var mf:metafile;i:integer);
(*--------------------------------------*)

var cl,ind:integer;

begin
verif(mf,2);
cl:=4;ind:=8;
parnbl(mf,cl,ind,i);
end;


procedure polfi(var mf:metafile;i:integer);
(*--------------------------------------*)
```

```
var cl,ind:integer;

begin
verif(mf,2);
cl:=4;ind:=9;
parnbl(mf,cl,ind,i);
end;


procedure initmf(num:integer;var mf:metafile);
(*--------------------------------------*)

begin
    ival:=1;f.nbre:=0;f.pos:=true;
    if num=2 then num:=32767
             else num:=19660;
    bmeta(mf);
    defvw(num,mf);
    set2d(mf);
end;


procedure endmf(var mf:metafile);
(*---------------------------*)

label 1;

begin
if b then goto 1;
if ival<>3 then
    begin
writeln(tty,'un appel a INITMF ou/et a BPICT ou/et a EPICT ');
writeln(tty,'n''a pas ete fait')
    end;
emeta(mf);
rempl(mf);
ival:=0;b:=false;
1:end;
```

2 EME PARTIE : ROUTINES D'APPLICATION.

```
procedure nuage(var mf:metafile;a,b:ndct2;dim,couleur,mk:
                integer;nom:achar);
(*-------------------------------------------------*)


var i:integer;

begin
    setcol(mf,couleur);
    setms(mf,mk);
    bpolym(mf);
    for i:=1 to dim do
```

```
            amove(mf,a[i],b[i]);
    epolym(mf);
    for i:=1 to dim do
        begin
            amove(mf,a[i],b[i]);
            outstr(mf,nom[i],5)
        end
end;


procedure polygo(var mf:metafile;a,b:ndct1;dim,couleur,lst,ist,
                 fc,fi:integer;nom:string10);
(*-------------------------------------------------------------*)

var i:integer;

begin
    setcol(mf,couleur);
    polist(mf,ist);
    setst(mf,lst);
    polest(mf,0);
    polfi(mf,fi);
    polfc(mf,fc);
    bpolyg(mf);
    amove(mf,a[1],b[1]);
    for i:=2 to dim do adraw(mf,a[i],b[i]);
    adraw(mf,a[1],b[1]);
    epolyg(mf);
    amove(mf,a[dim-1],b[dim-1]);
    outstr(mf,nom,10)

end;



function expo(k:integer):real;
(*-------------------------*)
var b:integer;
    tmp1:real;
begin
    tmp1:=1;
    if k>0 then for b:=1 to k do tmp1:=10*tmp1;
    if k<0 then for b:=1 to abs(k) do tmp1:= tmp1/10;
    expo:=tmp1
end;


procedure convrc(nbre : real; var char : string10);
(*-----------------------------------------------*)

label 1,2;
var i,j,k,tmp1 : integer;
    ival,min : real;

begin

min:=expo(-5);
```

```
for j:=1 to 10 do char[j]:=' ';
if abs(nbre) >= 10000 then goto 2;
j:=1;
(* 1 : trt du signe *)
if nbre < 0 then begin
                    char[1] := '-';
                    nbre := nbre * (-1)
                 end
else char[1] := ' ';
j:=2;
(* 2 : trt de la partie entiere      *)
(*   -- cas ou elle n'existe pas *)
if nbre < 1 then begin
                    char[j] := '0';
                    j := j+1;
                    goto 1
                 end;
(*  -- cas ou elle existe *)
if nbre < 10000 then i:=3;
if nbre < 1000 then i:=2;
if nbre < 100 then i:=1;
if nbre < 10 then i:=0;
for k:=i downto 0 do
    begin
        ival := nbre/expo(k);
        tmp1 := trunc(ival);
             case tmp1 of
                 0:char[j]:='0';
                 1:char[j]:='1';
                 2:char[j]:='2';
                 3:char[j]:='3';
                 4:char[j]:='4';
                 5:char[j]:='5';
                 6:char[j]:='6';
                 7:char[j]:='7';
                 8:char[j]:='8';
                 9:char[j]:='9';
             end;
         nbre := nbre-tmp1*expo(k);
         j := j+1;
    end;
(* 3 : trt de la partie decimale     *)
(*   -- cas ou elle n'existe pas *)
1:if nbre < min then begin
goto 2
                     end;
(*  -- cas ou elle existe *)
char[j]:='.';
j:=j+1;
while j <=5 do
    begin
        nbre := nbre*10;
        tmp1 := trunc(nbre);
        case tmp1 of
            0:char[j]:='0';
            1:char[j]:='1';
            2:char[j]:='2';
            3:char[j]:='3';
            4:char[j]:='4';
            5:char[j]:='5';
```

```
            6:char[j]:='6';
            7:char[j]:='7';
            8:char[j]:='8';
            9:char[j]:='9'
        end;
        nbre := nbre - tmp1;
        j:=j+1;
        if nbre < min then goto 2
    end;
2:end;



procedure compl1(var tab:ndct1;ptnb:integer);
(*-------------------------------------*)

var i:integer;

begin
for i:=1 to ptnb do
    tab[i]:=32767-tab[i];
end;



procedure compl2(var tab:ndct2;ptnb:integer);
(*-------------------------------------*)

var i:integer;

begin
for i:=1 to ptnb do
    tab[i]:=32767-tab[i];
end;


procedure cchar(tchar1:achar;ptnb:integer;var atab:rtab2;var
                tchar2:achar;var ftab:rtab1);
(*-----------------------------------------------------*)

var i : integer;

begin

for i:=1 to ptnb do
begin
    atab[i]:=i;ftab[i]:=i;
    tchar2[i]:=tchar1[i]
end;

end;



procedure fconv(frwt:rtab1;fptnb,ndc1,ndc2:integer;rw1,rw2:real;
                var fndct:ndct1);
(*-----------------------------------------------------------*)

var tmp2,tmp3,tmp4,step:real;
```

```
    tmp1,i:integer;

begin
(* 1 *)
tmp1:=ndc2-ndc1;
step:=rw2-rw1;
(* 2 *)
tmp2:=tmp1/step;
(* 3 *)
for i:=1 to fptnb do
    begin
        tmp3:=frwt[i]-rw1;
        tmp4:=tmp3*tmp2;
        fndct[i]:=round(tmp4)+ndc1
    end;

end;



procedure fconv2(frwt:rtab2;fptnb,ndc1,ndc2:integer;rw1,rw2:
                 real;var fndct:ndct2);
(*-------------------------------------------------------*)

var tmp2,tmp3,tmp4,step:real;
    tmp1,i:integer;

begin
(* 1 *)
tmp1:=ndc2-ndc1;
step:=rw2-rw1;
(* 2 *)
tmp2:=tmp1/step;
(* 3 *)
for i:=1 to fptnb do
    begin
        tmp3:=frwt[i]-rw1;
        tmp4:=tmp3*tmp2;
        fndct[i]:=round(tmp4)+ndc1
    end;
end;


procedure detpos(andct:ndct2;rwtab:rtab2;aptnb:integer;var pos:
                 integer;var err:boolean);
(*-------------------------------------------------------*)

var k:integer;

begin

    err:=false;
    k:=0;
                repeat k:=k+1
                until(abs(rwtab[k]-0)<0.0001)or(k>aptnb);
                if k<aptnb then pos:=andct[k]
                           else begin
            write(tty,'la grad. 0 manque sur l''axe ');
            err:=true
```

```
                                    end;
end;



procedure convan(rwtab:rtab2;dmax,aptnb:integer;var ndctab:ndct2
                ;var pos:integer;var err:boolean);
(*-----------------------------------------------------------*)

label 1;

var min,max:real;
    k,ndcm:integer;
    sw:boolean;

begin
sw:=false;
err:=false;min:=rwtab[1];
max:=rwtab[aptnb];
if min<0 then begin
                if max<0 then begin
                                  (* 1 *)
                                  ndcm:=31000-dmax-1000;
                                  ndctab[1]:=1000;
                                  pos:=31000-dmax
                              end
                else begin
                         (* 2 *)
                         ndcm:=31000;
                         ndctab[1]:=1000;
                         sw:=true
                     end
             end
else begin
        if min<0.0001 then begin
                               (* 3 *)
                               ndcm:=31000;
                               ndctab[1]:=1000+dmax;
                               pos:=1000+dmax
                           end
        else begin
                 (* 4 *)
                 ndcm:=31000;
                 ndctab[1]:=1000+dmax+1000;
                 pos:=1000+dmax
             end
    end;
fconv2(rwtab,aptnb,ndctab[1],ndcm,min,max,ndctab);
(* 5 *)
if sw then begin
            detpos(ndctab,rwtab,aptnb,pos,err);
            if err then goto 1;
            if pos<1000+dmax then begin
             ndctab[1]:=1000+dmax;ndcm:=31000;
             fconv2(rwtab,aptnb,ndctab[1],ndcm,min,max,ndctab);
             detpos(ndctab,rwtab,aptnb,pos,err);
                                  end;
```

```
            if pos>31000-dmax then begin
             ndctab[1]:=1000;ndcm:=31000-dmax;
             fconv2(rwtab,aptnb,ndctab[1],ndcm,min,max,ndctab);
             detpos(ndctab,rwtab,aptnb,pos,err);
                                    end
           end;

1:end;


procedure inv(nbptx,nbpty:integer;var res:boolean);
(*-----------------------------------------------*)

begin
    if nbptx<nbpty then res:=true
                   else res:=false
end;



procedure wclip(min,max:real;fptnb:integer;var f:rtabl;var err:
                boolean);
(*-------------------------------------------------------------*)

var tmp:real;
    i:integer;

begin

for i:=1 to fptnb do
    begin
        tmp:=f[i];
        if tmp<min then begin
                            err :=true;
                            f[i]:=min
                        end;
        if tmp>max then begin
                            err:=true;
                            f[i]:=max
                        end
    end;

end;



procedure contrt(a:arw;var f:frw;apt1,apt2,fptnb,typ:integer;
                 x,y:boolean;var err:boolean);
(*-------------------------------------------------------*)

var min,max:real;

begin

err:=false;
if not(x) then
if typ=1 then begin
                  min:=a.arwx[1];
```

```
                max:=a.arwx[apt1];
                wclip(min,max,fptnb,f.frwx,err);
            end;
if not(y) then begin
                min:=a.arwy[1];
                max:=a.arwy[apt2];
                wclip(min,max,fptnb,f.frwy,err)
          end
end;



procedure veropt(o:opt;var error:boolean);
(*-------------------------------------*)

label 1;
var k,total:integer;

begin
total:=0;
error:=false;
with o do begin
              if (not(cars in [1..2])) then cars:=1;
              if (not(acol in [1..7])) then acol:=7;
              if (not(ast in [1..4])) then ast :=4;
              if (not(znb in [1..40])) then znb:=1;
              if (not(axtyp in [1..2])) then axtyp:=1;
              if (not(fpronb in [0..50])) then fpronb:=0;
     if (not(axnb in [2..50]))
                  or (not(aynb in [2..50]))
                  or (fnb < 2) or (fnb >500)
          then begin
            writeln (tty,'valeur non permise pour une option.');
            error:=true;
            goto 1
               end;
            for k:=1 to znb do begin
                  with z[k] do begin
                    if (not(col in [1..7])) then col:=7;
                    if (not(lst in [1..4])) then lst:=4;
                    if (not(itype in [0..1]))then itype:=0;
                    if (not(fc in [1..7])) then fc:=7;
                    if (not(ist in [1..47])) then ist:=1;
                    if (dim <0) or (dim>500)
                    then begin
                         error:=true;
          writeln(tty,'nbre de points illegal dans une zone. ');
                         goto 1
                         end;
                               end;
                        total:=total+z[k].dim-1;
                              end;
              total:=total+1;
              if total <> fnb then begin
writeln(tty,'la decoupe de la fct en zone etant incorrecte,');
writeln(tty,'elle est ignoree.');
                                       znb:=1;
                                  end;
              if znb=1 then z[1].dim:=fnb;
```

```
          end;
l:end;



procedure verax(typ,axnb:integer;ax:rtab2;var error:boolean);
(*-------------------------------------------------------*)

label 1;
var k:integer;

begin

error:=false;
if typ=1 then begin
for k:=2 to axnb do
    if ax[k]<=ax[k-1] then begin
                              error:=true;
               writeln (tty,'grad de l''axe x non strct >.');
                              goto 1
                          end;
              end;
l:end;



procedure veray(aynb:integer;ay:rtab2;var error:boolean);
(*-------------------------------------------------*)

label 1;
var k:integer;

begin
error:=false;
for k:=2 to aynb do
    if ay[k]<=ay[k-1] then begin
                              error:=true;
               writeln (tty,'grad de l''axe y non strct >.');
                              goto 1
                          end;
l:end;



procedure verpro(pnbre,fnbre:integer;p:ndct2;var error:boolean);
(*-----------------------------------------------------------*)

label 1;
var k:integer;

begin
error:=false;
for k:=1 to pnbre do
    if (p[k]<=0) or (p[k]>fnbre)then begin
               writeln(tty,'numero de proj inexistant.');
                              error:=true
                                  end;
l:end;
```

```
procedure verpos(tab:rtab2;nb:integer;var err:boolean);
(*-------------------------------------------------*)

var i:integer;

begin
err:=false;
for i:=1 to nb do
    if tab[i] < 0 then err:=true;
end;



procedure convchar(aval:rtab2;aptnbr:integer;var tchar:achar);
(*-------------------------------------------------------*)

var i:integer;

begin

for i:=1 to aptnbr do
    convrc(aval[i],tchar[i]);

end;



procedure mefdh(tabh,tabv:ndct1;inf,nb,posh:integer;var ptabh,
                ptabv:ndct1);
(*-------------------------------------------------------*)

var k,ind:integer;

begin

ptabh[1]:=tabh[inf];
ptabv[1]:=posh;
ind:=inf-1;
nb:=nb+1;
for k:=2 to nb do
    begin
        ind:=ind+1;
        ptabh[k]:=tabh[ind];
        ptabv[k]:=tabv[ind]
    end;
nb:=nb+1;
ptabh[nb]:=tabh[ind];
ptabv[nb]:=posh;

end;



procedure mefdv(tabh,tabv:ndct1;inf,nb,posv:integer;var ptabh,
                ptabv:ndct1);
(*-------------------------------------------------------*)

var k,ind:integer;
```

```
begin

ptabh[1]:=posv;
ptabv[1]:=tabv[inf];
ind:=inf-1;
nb:=nb+1;
for k:=2 to nb do
    begin
        ind:=ind+1;
        ptabh[k]:=tabh[ind];
        ptabv[k]:=tabv[ind]
    end;
nb:=nb+1;
ptabh[nb]:=posv;
ptabv[nb]:=tabv[ind];

end;



procedure cgval(size : integer;var g:gval);
(*-------------------------------------*)

begin
if size = 1 then begin
                    with g do begin
                                    grsize:= 200;
                                    not1 := 1000;
                                    not2:= 1900;
                                    not3:= 2850;
                                    not4 := 3800;
                                    anam:= 4750;
                                    dmax:= 5750;
                                    nbpt:= 15
                                end
                 end
                else begin
                        with g do begin
                                    grsize:=200;
                                    not1 := 1600;
                                    not2:= 3000;
                                    not3:= 4200;
                                    not4 := 5400;
                                    anam:= 6800;
                                    dmax:=7600;
                                    nbpt:=11
                                end
                     end
end;



procedure text(size:integer;var mf:metafile);
(*-------------------------------------*)

var t1,t2:integer;
```

```
begin
if size =1 then begin
                    t1:=250;
                    t2:=450;
               end
          else begin
                    t1:=100;
                    t2:=1000;
               end;
setcsz(mf,t1,t2);
setcg(mf,32767,32767);
settft(mf,1);
settp(mf,1)
end;



procedure daxeh(ga:gval;andct:ndct2;posh,posv:integer;achart:
               achar;name:string10;color,style,ptnb:integer;
               var mf:metafile);
(*-----------------------------------------------------------*)
var tmp1,k,gvar,namvar,n1var,n2var:integer;
    p:boolean;

begin
(* 1 *)
setcol(mf,color);
setst(mf,style);
amove(mf,andct[1]-250,posh);
adraw(mf,andct[ptnb]+250,posh);
(* 2 *)
if posh>16383 then begin
                        gvar:=ga.grsize;
                        namvar:=ga.anam;
                        n1var:=ga.not1;
                        n2var:=ga.not2
                   end
              else begin
                        gvar:=-(ga.grsize);
                        namvar:=-(ga.anam);
                        n1var:=-(ga.not1);
                        n2var:=-(ga.not2)
                   end;
(* 3 *)
if posv < 16383 then amove(mf,27000,posh+namvar)
               else amove(mf,2000,posh+namvar);
outstr(mf,name,10);
(* 4 *)
for k:=1 to ptnb do begin
                        tmp1:=andct[k];
                        amove(mf,tmp1,posh);
                        adraw(mf,tmp1,posh+gvar)
                    end;
(* 5 *)
if ptnb < ga.nbpt then begin
                       for k:=1 to ptnb do begin
                            amove(mf,andct[k]-400,posh+n1var);
                            outstr(mf,achart[k],5)
                                        end
```

```
                end
            else begin
                    p:=true;
                    for k:=1 to ptnb do
                        begin
                          if p then begin
                                        tmp1:=n1var;
                                        p:=false
                                    end
                               else begin
                                        tmp1:=n2var;
                                        p:=true
                                    end;
                          amove(mf,andct[k]-400,posh+tmp1);
                          outstr(mf,achart[k],5)
                        end
             end;
end;



procedure daxev(ga:gval;andct:ndct2;posv,posh:integer;achart:
                achar;name:string10;color,style,ptnb:integer;
                var mf:metafile);
(*-------------------------------------------------------*)

var tmp1,k,gvar,namvar,n1var:integer;

begin
(* 1 *)
setcol(mf,color);
setst(mf,style);
amove(mf,posv,andct[1]-250);
adraw(mf,posv,andct[ptnb]+250);
(* 2 *)
if posv>16383 then begin
                       gvar:=500;
                       namvar:=1000;
                       n1var:=800
                   end
              else begin
                       gvar:=-500;
                       namvar:=-5000;
                       n1var:=-3000
                   end;
(* 3 *)
if posh<16383 then amove(mf,posv+namvar,32000)
              else amove(mf,posv+namvar,500);
outstr(mf,name,10);
(* 4 *)
for k:=1 to ptnb do begin
                       tmp1:=andct[k];
                       amove(mf,posv,tmp1);
                       adraw(mf,posv+gvar,tmp1)
                   end;
(* 5 *)
for k:=1 to ptnb do begin
   amove(mf,posv+n1var,andct[k]);
   outstr(mf,achart[k],5)
```

```
                    end;

end;



procedure pdraw(o:opt;f:fndc;posx:integer;var mf:metafile);
(*-----------------------------------------------------*)

var inf,i,t1,t2,t3,t4,t5,t6:integer;
    t7:string10;
    rndcx,rndcy:ndct1;

begin

inf:=1;
(* 1 *)
for i:=1 to o.znb do
    begin
        t1:=o.z[i].dim;
        t2:=o.z[i].col;
        t3:=o.z[i].lst;
        t4:=o.z[i].ist;
        t5:=o.z[i].fc;
        t6:=o.z[i].itype;
        t7:=o.z[i].nom;
(* 2 *)
    if rinv then mefdv(f.fndcy,f.fndcx,inf,t1,posx,rndcx,rndcy)
            else mefdh(f.fndcx,f.fndcy,inf,t1,posx,rndcx,rndcy);
(* 3 *)
        polygo(mf,rndcx,rndcy,t1+2,t2,t3,t6,t5,t4,t7);
        inf:=inf+t1-1
    end;
end;



procedure adraw2(ga:gval;acharx,achary:achar;var a:andc;posx,
                 posy:integer;o:opt;var mf:metafile);
(*-----------------------------------------------------*)

var t1:integer;

begin

if rinv then begin
    compl2(a.andcx,o.axnb);
    t1:=32767-posy;
    daxev(ga,a.andcx,posx,t1,acharx,o.axname,o.acol,o.ast,
          o.axnb,mf);
    daxeh(ga,a.andcy,t1,posx,achary,o.ayname,o.acol,o.ast,
          o.aynb,mf);
    end
else begin
    daxeh(ga,a.andcx,posx,posy,acharx,o.axname,o.acol,o.ast,
          o.axnb,mf);
    daxev(ga,a.andcy,posy,posx,achary,o.ayname,o.acol,o.ast,
```

```
            o.aynb,mf);
    end;

end;



procedure projc(projtab:ndct2;frwx,frwy:rtab1;fndcx,fndcy:ndct1;
                nb:integer;var rrwtx,rrwty:rtab2;var rndctx,
                rndcty:ndct2);
(*-------------------------------------------------------------*)

var i,ind:integer;

begin
for i:=1 to nb do
    begin
        ind:=projtab[i];
        rrwtx[i]:=frwx[ind];
        rrwty[i]:=frwy[ind];
        rndctx[i]:=fndcx[ind];
        rndcty[i]:=fndcy[ind]
    end
end;



procedure vvisf(ga:gval;rwf:rtab2;ndca:ndct2;ndcfh,ndcfv:ndct2;
                fpnb,vpos:integer;var mf:metafile);
(*-------------------------------------------------------------*)

label 1;
const prec=5;style=2;

var i,k,grvar,notvar:integer;
    char:string10;
    a,f,tmp:integer;

begin

(* 1 *)
if vpos>16283 then begin
                       grvar:=ga.grsize;
                       notvar:=1000
                   end
              else
                   begin
                       grvar:=-ga.grsize;
                       notvar:=-4500
                   end;
(* 2 *)
setst(mf,style);
for k:=1 to fpnb do
    begin
            amove(mf,ndcfh[k],ndcfv[k]);
            adraw(mf,vpos+grvar,ndcfv[k])
    end;
for k:=1 to fpnb do
    begin
```

```
i:=1;
f:=ndcfv[k];a:=ndca[i];
1:tmp:=abs(f-a);
(* a *)
if tmp > prec then begin
                   if not(rinv) then
                      (* b *)
                      if f<a then begin
                         convrc(rwf[k],char);
                         amove(mf,vpos+notvar,ndcfv[k]);
                         outstr(mf,char,5)
                                  end
                         else begin
                                  i:=i+1;
                                  a:=ndca[i];
                                  goto 1
                              end
                   else
                       (* c *)
                       if f>a then begin
                          convrc(rwf[k],char);
                          amove(mf,vpos+notvar,ndcfv[k]);
                          outstr(mf,char,5)
                                   end
                          else begin
                                  i:=i+1;
                                  a:=ndca[i];
                                  goto 1
                              end
    end
end
end;



procedure hvisf(ga:gval;rwf:rtab2;ndca:ndct2;ndcfh,ndcfv:ndct2;
                fpnb,hpos:integer;var mf:metafile);
(*-------------------------------------------------------------*)

label 1;
const prec=15;style=2;

var i,k,grvar,not3var,not4var:integer;
    char:string10;
    a,f,tmp:integer;
    p:boolean;

begin

p:=true;
if hpos>16283 then begin
                       grvar:=ga.grsize;
                       not3var:=ga.not3;
                       not4var:=ga.not4
                   end
              else
                   begin
                       grvar:=-ga.grsize;
                       not3var:=-ga.not3;
```

```
                        not4var:=-ga.not4
                    end;
setst(mf,style);
for k:=1 to fpnb do
    begin
         amove(mf,ndcfh[k],ndcfv[k]);
         adraw(mf,ndcfh[k],hpos+grvar)
    end;
for k:=1 to fpnb do
    begin
        i:=1;
        f:=ndcfh[k];a:=ndca[i];
        1:tmp:=abs(f-a);
        if tmp > prec then begin
                        if f<a then begin
                            if fpnb > ga.nbpt then
                                if p then begin
                            amove(mf,ndcfh[k]-400,hpos+not3var);
                            p:=false;
                                        end
                                    else begin
                            amove(mf,ndcfh[k]-400,hpos+not4var);
                            p:=true;
                                        end
                                    else
                            amove(mf,ndcfh[k]-400,hpos+not3var);
                            convrc(rwf[k],char);
                            outstr(mf,char,5)
                                end
                            else begin
                                    i:=i+1;
                                    a:=ndca[i];
                                    goto 1
                                end;
    end
    end
end;



procedure phrw(pt:rtab2;nbpt:integer;r:rtab2;var h:rtab2);
(*-------------------------------------------------*)

var i:integer;
    lrw:real;

begin
for i:=1 to nbpt-1 do begin
                        lrw:=pt[i+1]-pt[i];
                        h[i]:=r[i]/lrw
                    end;
end;



procedure minmax(tab:rtab2;dim:integer;var min,max:real;var
                 posmin,posmax:integer);
(*-------------------------------------------------------*)
```

```
var i:integer;

begin
max:=tab[1];min:=tab[1];
posmin:=1;posmax:=1;
for i:=2 to dim do
    if tab[i]>max then begin
                           max:=tab[i];
                           posmax:=i
                       end;
    if tab[i]<min then begin
                           min:=tab[i];
                           posmin:=i
                       end
end;


procedure phndc(hrw:rtab2;nbpt:integer;max:real;var h2:ndct2);
(*-------------------------------------------------------*)

const h=20000;
var ratio:real;
        i:integer;

begin
for i:=1 to nbpt-1 do
begin
    ratio:=hrw[i]/max;
    h2[i]:=round(h*ratio)
end
end;


procedure hdraw(ptndc,ndch:ndct2;ptnb:integer;o:opt;var
                mf:metafile);
(*-------------------------------------------------*)

var i,p1,p2,p3,p4,p5:integer;
    tabx,taby:ndct1;
    p6:string10;

begin
for i:=1 to ptnb-1 do begin
                   tabx[1]:=ptndc[i];
                   taby[1]:=5000;
                   tabx[2]:=tabx[1];
                   taby[2]:=ndch[i]+taby[1];
                   tabx[3]:=ptndc[i+1];
                   taby[3]:=taby[2];
                   tabx[4]:=tabx[3];
                   taby[4]:=taby[1];
                   p1:=o.z[i].col;
                   p2:=o.z[i].lst;
                   p3:=o.z[i].itype;
                   p4:=o.z[i].fc;
                   p5:=o.z[i].ist;
                   p6:=o.z[i].nom;
                   polygo(mf,tabx,taby,4,p1,p2,p3,p4,p5,p6)
```

```
                    end;
end;


procedure cercle(var fndcx,fndcy:ndctl;max:integer);
(*-----------------------------------------------*)

const vcst=100;vndc1=1000;vndc2=31000;rw1=-1;rw2=1;

var k,tmp:integer;
    frwx,frwy:rtabl;

begin
(* 1 *)
frwx[1]:=-1;frwx[max]:=-1;
frwy[1]:=0;frwy[max]:=0;
for k:=2 to vcst do
    begin
        tmp:=max-k+1;
        frwx[k]:=frwx[k-1]+0.02;
        frwy[k]:=sqrt(1-sqr(frwx[k]));
        frwx[tmp]:=frwx[k];
        frwy[tmp]:=-frwy[k]
    end;
frwx[101]:=1;frwy[101]:=0;
(* 2 *)
fconv(frwx,max,vndc1,vndc2,rw1,rw2,fndcx);
fconv(frwy,max,vndc1,vndc2,rw1,rw2,fndcy)
end;


procedure box(var mf:metafile);
(*-------------------------*)

type t = array[1..4]of integer;
var k:integer;
    x,y:t;

begin
x[1]:=0;y[1]:=0;
x[2]:=0;y[2]:=32767;
x[3]:=32767;y[3]:=32767;
x[4]:=32767;y[4]:=0;
amove(mf,x[1],y[1]);
for k:=2 to 4 do
    adraw(mf,x[k],y[k]);
adraw(mf,x[1],y[1])
end;


procedure constr(var tab:rtab2;nb:integer);
(*-------------------------------------*)

var k:integer;

begin
for k:=1 to nb do
```

```
tab[k]:=k
end;



procedure veracp(t1,t2:rtab2;vnb,inb:integer;o:opt;ccol,vcol,
                 icol:integer;var error:boolean);
(*-------------------------------------------------------*)

label 1;
var k:integer;
    tmp:real;

begin
error:=false;
if not(o.adrw) then begin
if(not(vnb in [1..50]))
  or (not(inb in [1..50])) then
      begin
          writeln(tty,'le nbre des var. ou des individus ');
          writeln(tty,'n''est pas compris entre 1 et 50');
          error:=true;
          goto 1
      end;
if not(o.cars in [1..2]) then o.cars:=1;
if not(o.ast in [1..4]) then o.ast:=4;
if not(o.acol in [1..7]) then o.acol:=7;
if not(ccol in [1..7]) then ccol:=7;
if not(vcol in [1..7]) then vcol:=7;
if not(icol in [1..7]) then icol:=7
                end;
for k:=1 to vnb do
    begin
    tmp:=t1[k];
    if(tmp < -1) or (tmp > 1) then begin
        writeln(tty,'une correlation a une valeur non comprise ');
        writeln(tty,'entre -1 et 1');
        error :=true;
        goto 1
                                  end;
    end;
for k:=1 to vnb do
    begin
    tmp:=t2[k];
    if(tmp < -1) or (tmp > 1) then begin
        writeln(tty,'une correlation a une valeur non comprise ');
        writeln(tty,'entre -1 et 1');
        error :=true;
        goto 1
                                   end;
    end;
1:end;



procedure acp(var1,var2,ind1,ind2:rtab2;vnb,indnb:integer;
              var o:opt;ccol,vcol,icol:integer;var mf:metafile);
(*-------------------------------------------------------------*)

label 1;
```

```
const vndc1=1000;vndc2=31000;rw1=-1;rw2=1;lst=4;ist=0;fc=1;fi=1;
      nom='          ';ms1=4;ms2=5;max=201;rinv=false;nb=50;

var arwx,arwy,tmp:rtab2;
    min1,min2,max1,max2:real;
    posmin,posmax,k:integer;
    av,ai:andc;
    err:boolean;

begin
(* 1 *)
if b then goto 1;
if ival<>2 then begin
    writeln(tty,'un appel a INITMF ou/et a BPICT n''a ');
    writeln(tty,'pas ete fait.');
    b:=true;
    goto 1
                end;
(* 2 *)
veracp(var1,var2,vnb,indnb,o,ccol,vcol,icol,err);if err then
   goto 1;
(* 3 *)
if not(o.adrw) then begin
          arwx[1]:=-1;arwy[1]:=-1;
          for k:=2 to 3 do begin
                              arwx[k]:=arwx[k-1]+1;
                              arwy[k]:=arwy[k-1]+1
                           end;
          o.axnb:=3;o.aynb:=3;
          cgval(o.cars,p.gv);
          text(o.cars,mf);
          convchar(arwx,o.axnb,p.tca);
          convan(arwx,p.gv.dmax,o.axnb,p.a.andcx,p.py,err);
          convan(arwy,p.gv.dmax,o.aynb,p.a.andcy,p.px,err)
                  end;
(* 4 *)
adraw2(p.gv,p.tca,p.tca,p.a,p.px,p.py,o,mf);
(* 5 *)
if not(o.adrw) then cercle(p.c.fndcx,p.c.fndcy,max);
(* 6 *)
polygo(mf,p.c.fndcx,p.c.fndcy,max,ccol,lst,ist,fc,fi,nom);
(* 7 *)
fconv2(var1,vnb,vndc1,vndc2,rw1,rw2,av.andcx);
fconv2(var2,vnb,vndc1,vndc2,rw1,rw2,av.andcy);
if not(o.adrw) then begin
                 constr(arwx,nb);
                 convchar(arwx,nb,p.tc)
              end;
nuage(mf,av.andcx,av.andcy,vnb,vcol,ms1,p.tc);
amove(mf,3000,2000);
outstr(mf,'0:variable ',10);
(* 8 *)
       (* a *)
minmax(ind1,indnb,min1,max1,posmin,posmax);
minmax(ind2,indnb,min2,max2,posmin,posmax);
tmp[1]:=abs(min1);tmp[2]:=abs(max1);tmp[3]:=abs(min2);
tmp[4]:=abs(max2);
minmax(tmp,lst,min1,max1,posmin,posmax);
       (* b *)
fconv2(ind1,indnb,vndc1,vndc2,-max1,max1,ai.andcx);
```

```
fconv2(ind2,indnb,vndc1,vndc2,-max1,max1,ai.andcy);
        (* c *)
nuage(mf,ai.andcx,ai.andcy,indnb,icol,ms2,p.tc);
        (* d *)
ind1[1]:=1;fconv2(ind1,rw2,vndc1,vndc2,-max1,max1,ai.andcx);
amove(mf,ai.andcx[1],16000);
adraw(mf,ai.andcx[1],15700);
outstr(mf,p.tc[1],5);
amove(mf,25000,2000);
outstr(mf,'X:individu',10);
box(mf);
1:end.



procedure fct(aw:arw;fw:frw;dchar:achar;p:ndct2;o:opt;
              var mf:metafile);
(*-------------------------------------------------*)

label 2;

var err:boolean;
    ad:andc;
    acharx,achary:achar;
    p1,p2,posx,posy:integer;
    ga:gval;
    prwx,prwy:rtab2;
    pndcx,pndcy:ndct2;
    p3,p4:real;

begin
(* 1 *)
if b then goto 2;
if ival<>2 then begin
    writeln(tty,'un appel a INITMF ou/et a BPICT n''a ');
    writeln(tty,'pas ete fait.');
    b:=true;
    goto 2
                end;
(* 2 *)
veropt(o,err);if err then goto 2;
if not(o.adrw) then begin
                    verax(o.axtyp,o.axnb,aw.arwx,err);
                    if err then goto 2;
                    veray(o.aynb,aw.arwy,err);
                    if err then goto 2;
                end;
verpro(o.fpronb,o.fnb,p,err);if err then goto 2;
contrt(aw,fw,o.axnb,o.aynb,o.fnb,o.axtyp,o.fxconv,o.fyconv,err);
if err then
writeln(tty,'attention,il y a eu clipping pour certaines ');
writeln(tty,'valeurs de la fct !');
(* 3 *)
cgval(o.cars,ga);
if not(o.adrw) then begin
if o.axtyp=1 then convchar(aw.arwx,o.axnb,acharx)
           else begin
                    cchar(dchar,o.axnb,aw.arwx,acharx,fw.frwx);
                    o.fnb:=o.axnb
                end;
```

```
text(o.cars,mf);
convan(aw.arwx,ga.dmax,o.axnb,ad.andcx,posy,err);
if err then goto 2;
convchar(aw.arwy,o.aynb,achary);
convan(aw.arwy,ga.dmax,o.aynb,ad.andcy,posx,err);
if err then goto 2;
inv(o.axnb,o.aynb,rinv);
text(o.cars,mf);
adraw2(ga,acharx,achary,ad,posx,posy,o,mf);
end;
(* 4 *)
if not (o.fxconv) then
     if rinv then begin
               p1:=32767-ad.andcx[1];
               p2:=32767-ad.andcx[2];
               p3:=aw.arwx[1];
               p4:=aw.arwx[2];
               fconv(fw.frwx,o.fnb,p1,p2,p3,p4,fd.fndcx);
               compll(fd.fndcx,o.fnb)
                   end
              else begin
               p1:=ad.andcx[1];
               p2:=ad.andcx[2];
               p3:=aw.arwx[1];
               p4:=aw.arwx[2];
               fconv(fw.frwx,o.fnb,p1,p2,p3,p4,fd.fndcx);
                   end;
if not (o.fyconv) then begin
                 p1:=ad.andcy[1];
                 p2:=ad.andcy[2];
                 p3:=aw.arwy[1];
                 p4:=aw.arwy[2];
                 fconv(fw.frwy,o.fnb,p1,p2,p3,p4,fd.fndcy)
                  end;
pdraw(o,fd,posx,mf);
(* 5 *)
if o.fpronb<>0 then begin
        projc(p,fw.frwx,fw.frwy,fd.fndcx,fd.fndcy,o.fpronb,
              prwx,prwy,pndcx,pndcy);
        if rinv then begin
           hvisf(ga,prwy,ad.andcy,pndcy,pndcx,o.fpronb,
                 32767-posy,mf);
           vvisf(ga,prwx,ad.andcx,pndcy,pndcx,o.fpronb,posx,mf)
                    end
                else begin
           hvisf(ga,prwx,ad.andcx,pndcx,pndcy,o.fpronb,posx,mf);
           vvisf(ga,prwy,ad.andcy,pndcx,pndcy,o.fpronb,posy,mf)
                                  end
                 end;
2:end;



procedure histo(aw:arw;o:opt;var mf:metafile);
(*----------------------------------------*)

label 1;
const rinv=false;

var axe,hndc,tab:ndct2;
```

```
    ga:gval;
    acharx:achar;
    i,tmp,posmin,posmax:integer;
    min,max:real;
    err:boolean;

begin


if b then goto 1;
(* 1 *)
if ival<>2 then begin
    writeln(tty,'un appel a INITMF ou/et a BPICT n''a ');
    writeln(tty,'pas ete fait.');
    b:=true;
    goto 1
                end;
o.aynb:=o.axnb;o.fnb:=o.axnb;o.znb:=o.axnb-1;
for i:= 1 to o.znb do o.z[i].dim:=2;
(* 2 *)
veropt(o,err);if err then goto 1;if o.axnb>41 then begin
    writeln(tty,'valeur non permise pour une option');
    goto 1
                                                    end;
verax(o.axtyp,o.axnb,aw.arwx,err);if err then goto 1;
verpos(aw.arwy,o.axnb-1,err);
if err then begin
        writeln(tty,'une repetition au moins est negative !');
        goto 1
            end;
(* 3 *)
tmp:=o.axnb;
convchar(aw.arwx,tmp,acharx);
cgval(o.cars,ga);text(o.cars,mf);
fconv2(aw.arwx,tmp,2000+ga.dmax,30000,aw.arwx[1],
       aw.arwx[tmp],axe);
daxeh(ga,axe,5000,0,acharx,o.axname,o.acol,o.ast,tmp,mf);
(* 4 *)
phrw(aw.arwx,tmp,aw.arwy,aw.arwx);
(* aw.arwx contient les hauteurs rw *)
aw.arwx[tmp]:=-1;
(* comme le nbre de hrw = tmp-1 il faut dc neutraliser le *)
(* dernier element de la table pour ne pas avoir d'ennui  *)
(* ds cmax  *)
minmax(aw.arwx,tmp,min,max,posmin,posmax);
phndc(aw.arwx,tmp,max,hndc);
hdraw(axe,hndc,tmp,o,mf);
(* 5 *)
for i:=1 to tmp do begin
                        tab[i]:=99999;
                        hndc[i]:=hndc[i]+5000
                   end;
vvisf(ga,aw.arwy,tab,axe,hndc,tmp-1,1000+ga.dmax,mf);
1:end;
```

PARTIE FORTRAN.

1 ERE PARTIE : REPRESENTATION ET MANIPULATION DE GRAPHES.

```
      integer function ADDARC(numarc,pntr,code,si,stt)
c     **********************
      integer numarc,pntr,code,si,stt

      integer gsarea(50,9),gaarea(250,9)
      integer nsom,narc
      integer nsc,nac
      common /area1/gsarea,gaarea
      common /area3/nsom,narc
      common /area4/nsc,nac

      integer result,dummy,GANUMI,GSNUMI

      result=0
      if(nac.gt.narc)goto 2000
      if((numarc.lt.0).or.(numarc.gt.999))goto 2000

      itmp1=GANUMI(numarc,dummy)
      if(itmp1.eq.0)goto 2000

      itmp1=GSNUMI(si,dummy)
      if(itmp1.eq.-1)goto 3000

      itmp1=GSNUMI(stt,dummy)
      if(itmp1.eq.-1)goto 3000

      do 1000 i=1,narc,1
      if((gaarea(i,8).eq.si).and.(gaarea(i,9).eq.stt))goto 3000
1000  continue

      call GADARC(numarc,pntr,code,si,stt)

      itmp2=GANUMI(numarc,ni)

      call jopen
      call PADARC(ni)
      call jclose

      goto 4000
2000  result=-1
      goto 4000
3000  result=-2
4000  ADDARC=result
      return
```

```
      end
c
c
c
      integer function ADDSOM(numsom,pntr,code,iline,icol)
c     **********************
      integer numsom,pntr,code,iline,icol

      integer nsom,narc
      integer nsc,nac
      common /area3/nsom,narc
      common /area4/nsc,nac

      integer result,dummy,GSNUMI,GPDECO

      result=0
      if(nsc.gt.nsom)goto 1000
      if((numsom.lt.0).or.(numsom.gt.99))goto 1000
      itmp1=GSNUMI(numsom,dummy)
      if(itmp1.eq.0)goto 1000

      itmp1=GPDECO(iline,icol)
      if(itmp1.ne.0)goto 2000

      call GADSOM(numsom,pntr,code,iline,icol)

      itmp2=GSNUMI(numsom,ni)

      call jopen
      call PADSOM(ni)
      call jclose

      goto 3000
1000  result=-1
      goto 3000
2000  result=-2
3000  ADDSOM=result
      return
      end
c
c
c
      integer function DELARC(numarc)
c     **********************
      integer numarc

      integer result,GANUMI

      result=0
      itmp1=GANUMI(numarc,ni)
      if(itmp1.eq.-1)goto 1000

      call jopen
      call PDLARC(ni)
      call jclose

      call GDLARC(ni)

      goto 2000
1000  result=-1
```

```
2000    DELARC=result
        return
        end
c
c
c
        integer function DELSOM(numsom)
c       **********************
        integer numsom

        integer result,GSNUMI

        result=0
        itmp1=GSNUMI(numsom,ni)
        if(itmp1.eq.-1)goto 1000

        call jopen
        call PDLSOM(ni)
        call jclose

        call GDLSOM(ni)

        goto 2000
1000    result=-1
2000    DELSOM=result
        return
        end
c
c
c
        integer function GLOAD(enss,ensa,ns,na)
c       **********************
        integer enss(ns,3),ensa(na,5),ns,na

        integer num1,numc
        common /area7/num1,numc

        integer result,i1(50),ic(50)

        result=0

        call GPENCO(num1,numc,i1,ic)

        do 1000 i=1,ns,1
        if(enss(i,1).lt.0)goto 1000
        itmp1=ADDSOM(enss(i,1),enss(i,2),enss(i,3),i1(i),ic(i))
        if(itmp1.ne.0)goto 3000
1000    continue

        do 2000 i=1,na,1
        itmp1=ADDARC(ensa(i,1),ensa(i,2),ensa(i,3),ensa(i,4),
        lensa(i,5))
        if(itmp1.ne.0)goto 4000
2000    continue

        goto 5000
3000    result=-1
        goto 5000
4000    result=-2
5000    GLOAD=result
```

```
      return
      end
c
c
c
      subroutine GOPEN(device,ori)
c     ****************
      integer device
      logical ori

c     Initialisation de la zone graphique

      call GINIT(ori)

c     Initialisation de l'image

      call PINIT(device)

      return
      end
c
c
c
      integer function GRAPNR(numarc,pntr)
c     ************************
      integer numarc,pntr

      integer gsarea(50,9),gaarea(250,9)
      common /area1/gsarea,gaarea

      integer result,GANUMI

      result=0
      itmp1=GANUMI(numarc,ni)
      if(itmp1.ne.0)goto 1000
      pntr=gaarea(ni,6)
      goto 2000
1000  result=-1
2000  GRAPNR=result

      return
      end
c
c
c
      integer function GRSPNR(numsom,pntr)
c     ************************
      integer numsom,pntr

      integer gsarea(50,9),gaarea(250,9)
      common /area1/gsarea,gaarea

      integer result,GSNUMI

      result=0
      itmp1=GSNUMI(numsom,ni)
      if(itmp1.ne.0)goto 1000
      pntr=gsarea(ni,6)
      goto 2000
1000  result=-1
```

```
2000    GRSPNR=result

        return
        end
c
c
c
        integer function GSAVE(enss,ensa,ns,na)
c       **********************
        integer enss(ns,3),ensa(na,5),ns,na

        integer gsarea(50,9),gaarea(250,9)
        integer nsom,narc
        integer numl,numc
        common /area1/gsarea,gaarea
        common /area3/nsom,narc
        common /area7/numl,numc

        integer result,t,code,il(50),ic(50)

        result=0
        if((ns.lt.1).or.(ns.gt.nsom))goto 3000
        if((na.lt.1).or.(na.gt.narc))goto 3000

        do 100 i=1,ns,1
100     enss(i,1)=-1

        call GPENCO(numl,numc,il,ic)

        do 1000 i=1,ns,1
        if(gsarea(i,7).eq.-1)goto 1000
        do 500 t=1,50,1
        if(gsarea(i,8).ne.il(t))goto 500
        if(gsarea(i,9).ne.ic(t))goto 500
        goto 600
500     continue
600     enss(t,1)=gsarea(i,5)
        enss(t,2)=gsarea(i,6)
        itmp1=gsarea(i,3)/16383
        itmp2=gsarea(i,4)/16383
        code=gsarea(i,1)*1000+gsarea(i,2)*100+itmp1*10+itmp2
        enss(t,3)=code
1000    continue

        t=1
        do 2000 i=1,narc,1
        if(gaarea(i,7).eq.-1)goto 2000
        ensa(t,1)=gaarea(i,5)
        ensa(t,2)=gaarea(i,6)
        itmp1=gaarea(i,3)/16383
        itmp2=gaarea(i,4)/16383
        code=gaarea(i,1)*1000+gaarea(i,2)*100+itmp1*10+itmp2
        ensa(t,3)=code
        ensa(t,4)=gaarea(i,8)
        ensa(t,5)=gaarea(i,9)
        t=t + 1
        if(t.gt.na)goto 4000
2000    continue

        goto 4000
```

```
3000    result=-1
4000    GSAVE=result
        return
        end
c
c
c
        integer function GWAPNR(numarc,pntr)
c       **********************
        integer numarc,pntr

        integer gsarea(50,9),gaarea(250,9)
        common /areal/gsarea,gaarea

        integer result,GANUMI

        result=0
        itmp1=GANUMI(numarc,ni)
        if(itmp1.ne.0)goto 1000
        gaarea(ni,6)=pntr
        goto 2000
1000    result=-1
2000    GWAPNR=result

        return
        end
c
c
c
        integer function GWSPNR(numsom,pntr)
c       **********************
        integer numsom,pntr
c
        integer gsarea(50,9),gaarea(250,9)
        common /areal/gsarea,gaarea

        integer result,GSNUMI

        result=0
        itmp1=GSNUMI(numsom,ni)
        if(itmp1.ne.0)goto 1000
        gsarea(ni,6)=pntr
        goto 2000
1000    result=-1
2000    GWSPNR=result

        return
        end
c
c
c
        integer function MDCARC(numarc,code)
c       **********************
        integer numarc,code

        integer result,GANUMI

        result=0
        itmp1=GANUMI(numarc,ni)
        if(itmp1.eq.-1)goto 1000
```

```
      call jopen
      call PDLARC(ni)
      call jclose

      call GMAARC(ni,code)

      call jopen
      call PADARC(ni)
      call jclose

      goto 3000
1000  result=-1
3000  MDCARC=result
      return
      end
c
c
c
      integer function MDCSOM(numsom,code)
c     **********************
      integer numsom,code

      integer gsarea(50,9),gaarea(250,9)
      integer backg
      common /area1/gsarea,gaarea
      common /area5/backg

      integer result,GSNUMI

      result=0
      itmp1=GSNUMI(numsom,ni)
      if(itmp1.eq.-1)goto 1000
      gsarea(ni,1)=backg

      call jopen
      call PADSOM(ni)
      call jclose

      call GMASOM(ni,code)

      call jopen
      call PADSOM(ni)
      call jclose

      goto 3000
1000  result=-1
3000  MDCSOM=result
      return
      end
c
c
c
      integer function MDNARC(numarc,newn)
c     **********************
      integer numarc,newn

      integer gsarea(50,9),gaarea(250,9)
      common /area1/gsarea,gaarea
```

```
      integer result,color,dummy,GANUMI

      result=0
      itmp1=GANUMI(numarc,ni)
      if(itmp1.eq.-1)goto 1000
      if((newn.lt.0).or.(newn.gt.999))goto 2000
      itmp2=GANUMI(newn,dummy)
      if(itmp2.eq.0)goto 2000

      color=gaarea(ni,1)

      call jopen
      call PDLARC(ni)
      call jclose

      gaarea(ni,1)=color

      call GMNARC(ni,newn)

      call jopen
      call PADARC(ni)
      call jclose

      goto 3000
1000  result=-1
      goto 3000
2000  result=-2
3000  MDNARC=result
      return
      end
c
c
c
      integer function MDNSOM(numsom,newn)
c     **********************
      integer numsom,newn

      integer gsarea(50,9),gaarea(250,9)
      integer backg

      common /area1/gsarea,gaarea
      common /area5/backg

      integer result,color,dummy,GSNUMI

      result=0
      itmp1=GSNUMI(numsom,ni)
      if(itmp1.eq.-1)goto 1000
      if((newn.lt.0).or.(newn.gt.99))goto 2000
      itmp2=GSNUMI(newn,dummy)
      if(itmp2.eq.0)goto 2000

      color=gsarea(ni,1)
      gsarea(ni,1)=backg

      call jopen
      call PADSOM(ni)
      call jclose

      gsarea(ni,1)=color
```

```
        call GMNSOM(ni,newn)

        call jopen
        call PADSOM(ni)
        call jclose
        goto 3000

1000    result=-1
        goto 3000
2000    result=-2
3000    MDNSOM=result
        return
        end
c
c
c
        integer function MDPARC(numarc,newsi,newst)
c       **********************
        integer numarc,newsi,newst

        integer gsarea(50,9),gaarea(250,9)
        integer nsom,narc
        common /area1/gsarea,gaarea
        common /area3/nsom,narc

        integer result,color,dummy,GSNUMI,GANUMI

        result=0
        itmp1=GANUMI(numarc,ni)
        if(itmp1.eq.-1)goto 1000
        itmp2=GSNUMI(newsi,dummy)
        if(itmp2.eq.-1)goto 2000
        itmp2=GSNUMI(newst,dummy)
        if(itmp2.eq.-1)goto 2000

        do 100 i=1,narc,1
        if((gaarea(i,8).eq.newsi).and.(gaarea(i,9).eq.newst))
        1goto 2000
100     continue

        color=gaarea(ni,1)

        call jopen
        call PDLARC(ni)
        call jclose

        gaarea(ni,1)=color

        call GMPARC(ni,newsi,newst)

        call jopen
        call PADARC(ni)
        call jclose

        goto 3000
1000    result=-1
        goto 3000
2000    result=-2
3000    MDPARC=result
        return
```

```
      end
c
c
c
      integer function MDPSOM(numsom,iline,icol)
c     **********************
      integer numsom,iline,icol
c
      integer gsarea(50,9),gaarea(250,9)
      integer nsom,narc
      integer backg
      common /area1/gsarea,gaarea
      common /area3/nsom,narc
      common /area5/backg

      integer result,color,GSNUMI,GPDECO

      result=0
      itmp1=GSNUMI(numsom,ni)
      if(itmp1.eq.-1)goto 5000
      itmp2=GPDECO(iline,icol)
      if(itmp2.ne.0)goto 6000


      color=gsarea(ni,1)
      gsarea(ni,1)=backg

      call jopen
      call PADSOM(ni)

      gsarea(ni,1)=color

      do 2000 i=1,narc,1
      if(gaarea(i,8).eq.gsarea(ni,5))goto 1000
      if(gaarea(i,9).eq.gsarea(ni,5))goto 1000
      goto 2000
1000  i1=i
      color=gaarea(i1,1)

c     Effacement des arcs incidents au sommet

      call PDLARC(i1)
      gaarea(i1,1)=color
2000  continue

      call jclose

      call GMPSOM(ni,iline,icol)

      call jopen
      call PADSOM(ni)

      do 4000 i=1,narc,1
      if(gaarea(i,8).eq.gsarea(ni,5))goto 3000
      if(gaarea(i,9).eq.gsarea(ni,5))goto 3000
      goto 4000
3000  i2=i
      call PADARC(i2)
4000  continue
```

```
      call jclose

      goto 7000
5000  result=-1
      goto 7000
6000  result=-2
7000  MDPSOM=result
      return
      end
c
c
c
      integer function MOVEGR(numgr)
c     **********************
      integer numgr

      real ginfo(8)
      common /area2/ginfo

      integer result

      result=0
      if((numgr.lt.1).or.(numgr.gt.6))goto 1000

      if(numgr.eq.1)call GMOVGR(2.,1.,ginfo(8))
      if(numgr.eq.2)call GMOVGR(2.,1.,-ginfo(8))
      if(numgr.eq.3)call GMOVGR(2.,-1.,-ginfo(8))
      if(numgr.eq.4)call GMOVGR(2.,-1.,ginfo(8))
      if(numgr.eq.5)call GMOVGR(2.,0.,0.)
      if(numgr.eq.6)call GMOVGR(1.,0.,0.)

      call PMOVGR

      goto 2000
1000  result=-1
2000  MOVEGR=result
      return
      end
c
c
c
      integer function PBACKG(color)
c     **********************
      integer color

      integer backg
      common /area5/backg

      integer result

      result=0
      if((color.lt.1).or.(color.gt.8))goto 1000

      call jbackg(color)
      call jframe

      backg=color

      goto 2000
1000  result=-1
```

```
2000    PBACKG=result
        return
        end
c
c
c
        subroutine PEND
c       ***************

        call jend

        return
        end
c
c
c
        subroutine PPRINT(mfname)
c       *****************
        double precision mfname

        call jmetaf(mfname)
        call jdinit(0)
        call jdevon(0)

        call PVALID

        call jdevof(0)
        call jdend(0)

        return
        end
c
c
c
        subroutine PVALID
c       *****************
        integer gsarea(50,9),gaarea(250,9)
        integer nsom,narc
        common /area1/gsarea,gaarea
        common /area3/nsom,narc

        integer status

        call GMOVGR(1.,0.,0.)

c       Destruction et recreation du segment retenu contenant
c       le graphe

        call jpurge(32000)
        call jropen(32000)

        do 1000 i=1,nsom,1
        status=gsarea(i,7)
        if(status.eq.0)goto 500
        if(status.eq.1)goto 800
        goto 1000
500     gsarea(i,7)=1
800     itmp1=i
        call PADSOM(itmp1)
1000    continue
```

```
      do 2000 i=1,narc,1
      status=gaarea(i,7)
      if(status.eq.0)goto 1500
      if(status.eq.1)goto 1800
      goto 2000
1500  gaarea(i,7)=1
1800  itmp1=i
      call PADARC(itmp1)
2000  continue

      call jrclos

      return
      end
c
c
c
      subroutine TTYINI(device)
c     *****************
      integer device

      decode(5,100,device)it1,it2
100   format(a3,o2)
      open(unit=25,device=device,access='seqinout')
      call stty(it2)

      return
      end
c
c
c
      subroutine GADARC(numarc,pntr,code,si,stt)
c     *****************
      integer numarc,pntr,code,si,stt

      integer gsarea(50,9),gaarea(250,9)
      integer nsom,narc
      integer nsc,nac
      common /area1/gsarea,gaarea
      common /area3/nsom,narc
      common /area4/nsc,nac

      do 1000 i=1,narc,1
      if(gaarea(i,7).eq.-1)goto 1500
1000  continue
1500  itmp2=i

      call GMAARC(itmp2,code)
      call GMNARC(itmp2,numarc)

      gaarea(itmp2,6)=pntr
      gaarea(itmp2,7)=0

      call GMPARC(itmp2,si,stt)
      nac=nac+1

      return
      end
c
```

```
c
c
      subroutine GADSOM(numsom,pntr,code,iline,icol)
c     *****************
      integer numsom,pntr,code,iline,icol

      integer gsarea(50,9),gaarea(250,9)
      integer nsom,narc
      integer nsc,nac
      common /area1/gsarea,gaarea
      common /area3/nsom,narc
      common /area4/nsc,nac

      do 1000 i=1,nsom,1
      if(gsarea(i,7).eq.-1)goto 1500
1000  continue
1500  itmp2=i

      call GMASOM(itmp2,code)

      gsarea(itmp2,5)=numsom
      gsarea(itmp2,6)=pntr
      gsarea(itmp2,7)=0

      call GMPSOM(itmp2,iline,icol)
      nsc=nsc+1

      return
      end
c
c
c
      subroutine GDLARC(ninarc)
c     *****************
      integer ninarc

      integer gsarea(50,9),gaarea(250,9)
      integer nsc,nac
      common /area1/gsarea,gaarea
      common /area4/nsc,nac

      do 1000 i=1,9,1
1000  gaarea(ninarc,i)=-1

      nac=nac-1
      return
      end
c
c
c
      subroutine GDLSOM(ninsom)
c     *****************
      integer ninsom

      integer gsarea(50,9),gaarea(250,9)
      integer nsom,narc
      integer nsc,nac
      common /area1/gsarea,gaarea
      common /area3/nsom,narc
      common /area4/nsc,nac
```

```
      numsom=gsarea(ninsom,5)

      do 100 i=1,9,1
100   gsarea(ninsom,i)=-1

      do 2000 i=1,narc,1
      if(gaarea(i,8).eq.numsom)goto 1000
      if(gaarea(i,9).eq.numsom)goto 1000
      goto 2000
1000  num=i

c     Destruction des arcs incidents au sommet

      call GDLARC(num)
2000  continue

      nsc=nsc-1
      return
      end
c
c
c
      subroutine GINIT(ori)
c     ****************
      logical ori

      integer gsarea(50,9),gaarea(250,9)
      integer nsom,narc
      integer nsc,nac
      logical orient
      common /area1/gsarea,gaarea
      common /area3/nsom,narc
      common /area4/nsc,nac
      common /area6/orient

      nsom=50
      narc=250
      orient=ori

      do 1000 i=1,nsom,1
      do 1000 j=1,9,1
      gsarea(i,j)=-1
1000  continue

      do 2000 i=1,narc,1
      do 2000 j=1,9,1
      gaarea(i,j)=-1
2000  continue

      nsc=1
      nac=1

      return
      end
c
c
c
      subroutine GMAARC(ninarc,code)
c     *****************
```

```
      integer ninarc,code

      integer gsarea(50,9),gaarea(250,9)
      common /areal/gsarea,gaarea

      integer cl,st,lt,br

      call GADECO(code,cl,st,lt,br)

      gaarea(ninarc,1)=cl
      gaarea(ninarc,2)=st
      gaarea(ninarc,3)=lt
      gaarea(ninarc,4)=br

      return
      end
c
c
c
      subroutine GMASOM(ninsom,code)
c     *****************
      integer ninsom,code

      integer gsarea(50,9),gaarea(250,9)
      common /areal/gsarea,gaarea

      integer cl,st,lt,br

      call GADECO(code,cl,st,lt,br)

      gsarea(ninsom,1)=cl
      gsarea(ninsom,2)=st
      gsarea(ninsom,3)=lt
      gsarea(ninsom,4)=br

      return
      end
c
c
c
      subroutine GMNARC(oldni,newn)
c     *****************
      integer oldni,newn

      integer gsarea(50,9),gaarea(250,9)
      common /areal/gsarea,gaarea

      gaarea(oldni,5)=newn

      return
      end
c
c
c
      subroutine GMNSOM(oldni,newn)
c     *****************
      integer oldni,newn

      integer gsarea(50,9),gaarea(250,9)
      integer nsom,narc
```

```
      common /area1/gsarea,gaarea
      common /area3/nsom,narc

      itmp2=gsarea(oldni,5)

      do 1000 i=1,narc,1
      if(gaarea(i,8).eq.itmp2)gaarea(i,8)=newn
      if(gaarea(i,9).eq.itmp2)gaarea(i,9)=newn
1000  continue
      gsarea(oldni,5)=newn

      return
      end
c
c
c
      subroutine GMPARC(ninarc,newsi,newst)
c     *****************
      integer ninarc,newsi,newst

      integer gsarea(50,9),gaarea(250,9)
      common /area1/gsarea,gaarea

      gaarea(ninarc,8)=newsi
      gaarea(ninarc,9)=newst

      return
      end
c
c
c
      subroutine GMPSOM(ninsom,iline,icol)
c     *****************
      integer ninsom,iline,icol

      integer gsarea(50,9),gaarea(250,9)
      common /area1/gsarea,gaarea

      gsarea(ninsom,8)=iline
      gsarea(ninsom,9)=icol

      return
      end
c
c
c
      subroutine PADARC(ninarc)
c     *****************
      integer ninarc

      integer gsarea(50,9),gaarea(250,9)
      real ginfo(8)
      logical orient
      common /area2/ginfo
      common /area1/gsarea,gaarea
      common /area6/orient

      integer cl,st,lt,br,si,stt,string,GANUMI,GSNUMI

      cl=gaarea(ninarc,1)
```

```
      st=gaarea(ninarc,2)
      lt=gaarea(ninarc,3)
      br=gaarea(ninarc,4)

      call jcolor(cl)
      call jlstyl(st)
      call jlwide(lt)
      call jinten(br)

      si =gaarea(ninarc,8)
      stt=gaarea(ninarc,9)
      scalfa=ginfo(5)

      if(si.ne.stt)goto 500

c     CAS 0 : Trace d'une boucle

      itmpl=GSNUMI(si,nisi)
      call PCONVP(nisi,xl,yl)
      x00=xl + 2.*scalfa
      y00=yl + 2.*scalfa

      radius=scalfa*SQRT(2.)
      nseg=1

      call jcircl(x00,y00,0.,radius,nseg)
      call jsize(1.*scalfa,0.8*scalfa)
      call jpath(1)
      call jjust(2,2)

      goto 7000

500   itmpl=GSNUMI(si,nisi)
      itmpl=GSNUMI(stt,nist)
      call PCONVP(nisi,xl,yl)
      call PCONVP(nist,x2,y2)

      radtod=45./ATAN(1.0)

      nseg=1
      radius=1.5*scalfa

      call jsize(1.*scalfa,0.8*scalfa)

      if(xl.ne.x2)goto 2000
      if(yl.lt.y2)goto 1000

c     CAS 1 : Les sommets initial et terminal sont alignes
c             verticalement, le sommet terminal est vers le
c             bas

      xpl=xl + scalfa;ypl=yl - scalfa
      xp2=x2 + scalfa;yp2=y2 + scalfa
      x00=(3*xp2 + xpl)/4.
      y00=(3*yp2 + ypl)/4.

      call jmove(xpl,ypl)
      call jdraw(xp2,yp2)
      call jjust(1,2)
      call jpath(2)
```

```
        if(orient)goto 6200
        goto 7000

c       CAS 2 : Les sommets initial et terminal sont alignes
c               verticalement, le sommet terminal est vers le
c               haut.
c

1000    xp1=x1 - scalfa;yp1=y1 + scalfa
        xp2=x2 - scalfa;yp2=y2 - scalfa
        x00=(3*xp2 + xp1)/4.
        y00=(3*yp2 + yp1)/4.

        call jmove(xp1,yp1)
        call jdraw(xp2,yp2)
        call jjust(3,2)
        call jpath(2)

        if(orient)goto 6300
        goto 7000

2000    if(x1.lt.x2)goto 3000

c       CAS 3 : Les sommets initial et terminal sont disposes de
c               facon quelconque, le sommet terminal est le plus
c               a gauche.

        xp1=x1 - scalfa;yp1=y1 - scalfa
        xp2=x2 + scalfa;yp2=y2 - scalfa
        x00=(3*xp2 + xp1)/4.
        y00=(3*yp2 + yp1)/4.

        call jmove(xp1,yp1)
        call jdraw(xp2,yp2)
        call jjust(2,3)
        call jpath(1)

        if(orient)goto 2100
        goto 7000

2100    if(y1.le.y2)goto 2200
        angle=0.
        goto 6100
2200    angle=360.
        goto 6100

c       CAS 4 : Les sommets initial et terminal sont disposes de
c               facon quelconque, le sommet terminal est le plus
c               a droite.

3000    xp1=x1 + scalfa;yp1=y1 + scalfa
        xp2=x2 - scalfa;yp2=y2 + scalfa
        x00=(3*xp2 + xp1)/4.
        y00=(3*yp2 + yp1)/4.

        call jmove(xp1,yp1)
        call jdraw(xp2,yp2)
        call jjust(2,1)
        call jpath(1)
```

```
        if(orient)goto 6000
        goto 7000

6000    angle=180.
6100    talpha=(y2 - y1)/(x2 - x1)
        alpha =ATAN(talpha)*radtod
        alphap=alpha + angle
        goto 6600
6200    alphap=90.
        goto 6600
6300    alphap=270.
        goto 6600
6600    a0=alphap - 15.
        a1=alphap + 15.

        x01=xp1 + (xp2 - xp1)/2.
        y01=yp1 + (yp2 - yp1)/2.
        call jsectr(x01,y01,0.,radius,nseg,a0,a1)
7000    call jmove(x00,y00)

        num=gaarea(ninarc,5)
        call GENCOD(num,1,string)

        call jxtext(1,string)

        return
        end
c
c
c
        subroutine PADSOM(ninsom)
c       *****************
        integer ninsom

        real ginfo(8)
        integer gsarea(50,9),gaarea(250,9)
        common /area2/ginfo
        common /area1/gsarea,gaarea

        real dx(4),dy(4)
        integer cl,st,lt,br,string,GSNUMI

        cl=gsarea(ninsom,1)
        st=gsarea(ninsom,2)
        lt=gsarea(ninsom,3)
        br=gsarea(ninsom,4)

        call jcolor(cl)
        call jlstyl(st)
        call jlwide(lt)
        call jinten(br)

        call PCONVP(ninsom,x,y)

        scalfa=ginfo(5)

        call jmove(x-scalfa,y-scalfa)

        dx(1)=0.
```

```
      dy(1)=2.*scalfa
      dx(2)=2.*scalfa
      dy(2)=0.
      dx(3)=0.
      dy(3)=-2.*scalfa
      dx(4)=-2.*scalfa
      dy(4)=0.

      call jrpoly(dx,dy,4)

      num=gsarea(ninsom,5)
      call GENCOD(num,1,string)

      call jpath(1)
      call jsize(1.*scalfa,0.8*scalfa)
      call jjust(2,2)
      call jmove(x,y)
      call jxtext(1,string)

      return
      end
c
c
c
      subroutine PDLARC(ninarc)
c     *****************
      integer ninarc

      integer gsarea(50,9),gaarea(250,9)
      integer backg
      common /area1/gsarea,gaarea
      common /area5/backg

      gaarea(ninarc,1)=backg

      call PADARC(ninarc)

      return
      end
c
c
c
      subroutine PDLSOM(ninsom)
c     *****************
      integer ninsom

      integer gsarea(50,9),gaarea(250,9)
      integer nsom,narc
      integer backg
      common /area1/gsarea,gaarea
      common /area3/nsom,narc
      common /area5/backg

      gsarea(ninsom,1)=backg

      call PADSOM(ninsom)

      do 2000 i=1,narc,1
      if(gaarea(i,8).eq.gsarea(ninsom,5))goto 1000
      if(gaarea(i,9).eq.gsarea(ninsom,5))goto 1000
```

```
      goto 2000
1000  itmp1=i

c     Destruction des arcs incidents au sommet

      call PDLARC(itmp1)
2000  continue

      return
      end
c
c
c
      subroutine PGRILL(nl,nc)
c     ************************
      integer nl,nc

      real xv(4),yv(4),zv(4)
      integer string

      call jropen(1)

      call jlwide(0)
      call jinten(0)
      call jiwind(xv,yv,zv)
      call jmove(xv(4),yv(4))
      call jpoly(xv,yv,4)

      call jsize(0.25,0.50)
      call jjust(2,3)

      do 1000 i=1,nc,1
      x0=float(i)
      call jmove(x0,yv(1))
      call jdraw(x0,yv(1) - 0.20)
      call jmove(x0,yv(1) - 0.25)
      num = i
      call GENCOD(num,1,string)
      call j2text(1,string)
1000  continue
      call jjust(2,1)

      do 2000 i=1,nc,1
      x0=float(i)
      call jmove(x0,yv(2))
      call jdraw(x0,yv(2) + 0.20)
      call jmove(x0,yv(2) + 0.25)
      num = i
      call GENCOD(num,1,string)
      call j2text(1,string)
2000  continue
      call jjust(3,2)

      do 3000 i=1,nl,1
      y0=float(i)
      call jmove(xv(1),y0)
      call jdraw(xv(1) - 0.20,y0)
      call jmove(xv(1) - 0.25,y0)
      num = i
      call GENCOD(num,1,string)
```

```
      call j2text(1,string)
3000  continue
      call jjust(1,2)

      do 4000 i=1,nl,1
      y0=float(i)
      call jmove(xv(3),y0)
      call jdraw(xv(3) + 0.20,y0)
      call jmove(xv(3) + 0.25,y0)
      num = i
      call GENCOD(num,1,string)
      call j2text(1,string)
4000  continue

      call jrclos

      return
      end
c
c
c
      subroutine PINIT(device)
c     ****************
      integer device

      integer numl,numc
      integer backg
      real ginfo(8)
      common /area2/ginfo
      common /area5/backg
      common /area7/numl,numc

      call jbegin
      call jdinit(device)
      call jdevon(device)
      call jaspek(device,ratio)
      call jvspac(-1.,1.,-ratio,ratio)
      call jvport(-0.9,0.9,-0.9*ratio,0.9*ratio)
      call jfiles(1,0,1)
      call jqualt(3)
      call jttype(2)

      numl=9
      numc=11
      backg=8
      ginfo(1)=72.
      ginfo(2)=72.*ratio
      ginfo(3)=float(numc) + 1.
      ginfo(4)=float(numl) + 1.
      ginfo(5)=1.
      ginfo(6)=0.
      ginfo(7)=0.
      ginfo(8)=ratio

      call jwindo(0.,ginfo(3),0.,ginfo(4))
      call PGRILL(numl,numc)
      call jwindo(0.,ginfo(1),0.,ginfo(2))

c     Creation initiale du segment contenant le graphe
```

```
      call jropen(32000)
      call jrclos

      return
      end
c
c
c
      subroutine PMOVGR
c     *****************
      real ginfo(8)
      common /area2/ginfo

      dummy=0.0
      scalfa=ginfo(5)
      dx=ginfo(6)
      dy=ginfo(7)

      call jt2all(1,0.,0.,scalfa,scalfa,dummy,dx,dy)
      call jt2all(32000,0.,0.,scalfa,scalfa,dummy,dx,dy)

      return
      end
c
c
c
      subroutine GADECO(code,cl,st,lt,br)
c     *****************
      integer code,cl,st,lt,br

      integer backg
      common /area5/backg

      itmp1=code
      cl=IFIX(itmp1/1000.)

      itmp2=MOD(itmp1,1000)
      st=IFIX(itmp2/100.)

      itmp3=MOD(itmp2,100)
      lt=IFIX(itmp3/10.)

      br=MOD(itmp3,10)

      if((cl.lt.1).or.(cl.gt.8))goto 4000
1000  if((st.lt.1).or.(st.gt.4))st=4
2000  if((lt.lt.1).or.(lt.gt.3))goto 5000
      lt=(lt - 1)*16383
3000  if((br.lt.1).or.(br.gt.3))goto 6000
      br=(br - 1)*16383
      goto 7000

4000  do 4500 i=1,8,1
      if(i.ne.backg)goto 4600
4500  continue
4600  cl=i
      goto 1000
5000  lt=16383
      goto 3000
6000  br=16383
```

```
7000    return
        end
c
c
c
        integer function GANUMI(numarc,ninarc)
c       **********************
        integer numarc,ninarc

        integer gsarea(50,9),gaarea(250,9)
        integer nsom,narc
        common /area1/gsarea,gaarea
        common /area3/nsom,narc

        integer result,i

        result=0
        ninarc=0
        do 1000 i=1,narc,1
        if(gaarea(i,5).ne.numarc)goto 1000
        ninarc=i
        goto 2000
1000    continue
        result=-1
2000    GANUMI=result
        return
        end
c
c
c
        subroutine GENCOD(num,l,string)
c       *****************
        integer num,l,string

        l=0
        string='   '
        if((num.lt.0).or.(num.gt.999))return
        if(num.le.9)goto 100
        if((num.gt.9).and.(num.le.99))goto 200
        if(num.gt.99)goto 300
1       format(i1)
2       format(i2)
3       format(i3)
100     encode(1,1,string)num
        l=1
        return
200     encode(2,2,string)num
        l=2
        return
300     encode(3,3,string)num
        l=3
        return
        end
c
c
c
        subroutine GMOVGR(scale,tx,ty)
c       *****************
        real scale,tx,ty
```

```
      real ginfo(8)
      common /area2/ginfo

      ginfo(5)=scale
      ginfo(6)=tx
      ginfo(7)=ty

      return
      end
c
c
c
      integer function GPDECO(iline,icol)
c     **********************
      integer iline,icol

      integer num1,numc
      integer gsarea(50,9),gaarea(250,9)
      integer nsom,narc
      common /area1/gsarea,gaarea
      common /area3/nsom,narc
      common /area7/num1,numc

      integer result,reste1,reste2

      result=0

      if((iline.lt.1).or.(iline.gt.num1))goto 2000
      if((icol.lt.1).or.(icol.gt.numc))goto 2000

      reste1=MOD(iline,2)
      reste2=MOD(icol,2)

      if((reste1.eq.0).and.(reste2.eq.0))goto 100
      if((reste1.ne.0).and.(reste2.ne.0))goto 100
      goto 2000

100   do 1000 i=1,nsom,1
      if(gsarea(i,8).ne.iline)goto 1000
      if(gsarea(i,9).ne.icol )goto 1000
      goto 3000
1000  continue
      goto 4000
2000  result=-1
      goto 4000
3000  result=-2
4000  GPDECO=result
      return
      end
c
c
c
      subroutine GPENCO(n1,nc,il,ic)
c     ******************
      integer n1,nc
      integer il(1),ic(1)

      icol=0
      ilin=n1
```

```
      k=1
1000  icol=icol+1
      if(icol.gt.nc)goto 4000
      itmp1=MOD(icol,2)
      if(itmp1.eq.0)goto 2000
1500  if(ilin.ge.1)goto 3000
      ilin=nl-1
      goto 1000
2000  if(ilin.ge.2)goto 3500
      ilin=nl
      goto 1000
3000  il(k)=ilin
      ic(k)=icol
      ilin=ilin-2
      k=k+1
      goto 1500
3500  il(k)=ilin
      ic(k)=icol
      ilin=ilin-2
      k=k+1
      goto 2000
4000  return
      end
c
c
c
      integer function GSNUMI(numsom,ninsom)
c     **********************
      integer numsom,ninsom

      integer gsarea(50,9),gaarea(250,9)
      integer nsom,narc
      common /area1/gsarea,gaarea
      common /area3/nsom,narc

      integer result

      result=0
      ninsom=0
      do 1000 i=1,nsom,1
      if(gsarea(i,5).ne.numsom)goto 1000
      ninsom=i
      goto 2000
1000  continue
      result=-1
2000  GSNUMI=result
      return
      end
c
c
c
      subroutine PCONVP(ninsom,xpos,ypos)
c     *****************
      integer ninsom
      real xpos,ypos

      real ginfo(8)
      integer gsarea(50,9),gaarea(250,9)
      common /area2/ginfo
      common /area1/gsarea,gaarea
```

```
      integer il,ic
      real nux,nuy

      xsize= ginfo(1)
      ysize= ginfo(2)
      nux=   ginfo(3)
      nuy=   ginfo(4)
      scalfa=ginfo(5)
      dx=    ginfo(6)
      dy=    ginfo(7)
      il=    gsarea(ninsom,8)
      ic=    gsarea(ninsom,9)
      x=     (ic*xsize)/nux
      y=     (il*ysize)/nuy

      call jconwv(x,y,0.,vx,vy)

      vx = vx*scalfa + dx
      vy = vy*scalfa + dy

      call jconvw(vx,vy,x,y,z)

      xpos = x
      ypos = y

      return
      end
c
c
c
```

2 EME PARTIE : REPRESENTATION DE CLASSIFICATIONS HIERARCHIQUES ET D'HISTOGRAMMES.

```
      subroutine DINIT(device)
c     ****************
      integer device

      integer backg
      common /area5/backg

      call jbegin
      call jdinit(device)
      call jdevon(device)
      call jaspek(device,ratio)
      call jvspac(-1.,1.,-ratio,ratio)
      call jvport(-0.8,0.8,-0.8*ratio,0.8*ratio)
      call jfiles(1,0,1)
      call jqualt(2)

      backg=8
```

```
      return
      end
c
c
c
      subroutine DPRINT(mfname)
c     *****************
      double precision mfname

      integer nci,gx(40),gy(39),dumi(2),axc,dir,nd,nda,pcc
      integer pcn(2)
      real xs,ys,abi(40),ori(39),yp(39),xg(39),xd(39),yg(39)
      real yd(39)
      common /area10/nci,gx,gy,dumi,axc,dir,nd,nda,pcc,pcn
      common /area11/xs,ys,abi,ori,yp,xg,xd,yg,yd

      call jmetaf(mfname)
      call jdinit(0)
      call jdevon(0)

      call jwindo(0.0,xs,0.0,ys)

      call DAXEH(abi,nci,gx,dumi,axc,dir,2,xs,ys)
      call DAXEV(ori,nd,gy,dumi,axc,1,5,xs,ys)
      call DRAW(nda,yp,xg,xd,yg,yd,pcc)
      call TITRE(pcn,axc,1,xs,ys)

      call jdevof(0)
      call jdend(0)

      return
      end
c
c
c
      integer function HISTO(rep,lim,n,ind,ecode,icode,axname,
     1                       axcode,pcname)
c     **********************
      integer n,rep(n),ind(n),ecode(n),icode(n),axname(2)
      integer axcode,pcname(2)
      real lim(1)

      integer ni,nl,indi(40),ec(40),ic(40),axn(2),axc,dir,pcn(2)
      integer xnom(41),r(40)
      real li(41),xs,ys,xpos(41),ypos(40)
      common /area8/r,ni,nl,indi,ec,ic,axn,axc,dir,pcn,xnom
      common /area9/li,xs,ys,xpos,ypos

      real h(40),absi(41)
      integer dirg,grd(41)

      result=0

c     Verification des contraintes

      if((n.lt.1).or.(n.gt.40))goto 2000

      do 1000 i=1,n,1
      if(rep(i).le.0)goto 3000
      if(lim(i + 1).le.lim(i))goto 4000
```

```
1000    continue

c       Trace de l'histogramme

        xsize=100.
        ysize=60.
        dirg=1
        nlim=n + 1

        call HCALC(rep,lim,n,h)

        call jwindo(lim(1),lim(nlim),0.0,1.0)

        call HDRAW(n,lim,h,ind,ecode,icode)
        call HPROJ(rep,lim,h,n,lim(nlim)-lim(1),1.0)
        call HCONV(lim,nlim,xsize,absi,grd)

        call jwindo(0.0,xsize,0.0,ysize)

        if(nlim.gt.10)dirg=2
        call DAXEH(absi,nlim,grd,axname,axcode,dirg,5,xsize,ysize)
        call TITRE(pcname,axcode,1,xsize,ysize)

c       Memorisation des donnees pour le trace dans un
c       metafichier

        ni=n
        nl=nlim
        axc=axcode
        dir=dirg
        xs=xsize
        ys=ysize

        do 1400 i=1,2,1
        axn(i)=axname(i)
1400    pcn(i)=pcname(i)

        do 1500 i=1,n,1
        indi(i)=ind(i)
        ec(i)=ecode(i)
        ic(i)=icode(i)
        r(i)=rep(i)
1500    ypos(i)=h(i)

        do 1600 i=1,nlim,1
        xnom(i)=grd(i)
        li(i)=lim(i)
1600    xpos(i)=absi(i)

        goto 5000
2000    result=-1
        goto 5000
3000    result=-2
        goto 5000
4000    result=-3
5000    HISTO=result
        return
        end
c
c
```

```
c
      subroutine HPRINT(mfname)
c     *****************
      double precision mfname

      integer ni,nl,indi(40),ec(40),ic(40),axn(2),axc,dir
      integer xnom(41),r(40),pcn(2)
      real li(41),xs,ys,xpos(41),ypos(40)
      common /area8/r,ni,nl,indi,ec,ic,axn,axc,dir,pcn,xnom
      common /area9/li,xs,ys,xpos,ypos

      call jmetaf(mfname)
      call jdinit(0)
      call jdevon(0)

      call jwindo(li(1),li(nl),0.0,1.0)

      call HDRAW(ni,li,ypos,indi,ec,ic)
      call HPROJ(r,li,ypos,ni,li(nl)-li(1),1.0)

      call jwindo(0.0,xs,0.0,ys)

      call DAXEH(xpos,nl,xnom,axn,axc,dir,5,xs,ys)
      call TITRE(pcn,axc,1,xs,ys)

      call jdevof(0)
      call jdend(0)

      return
      end
c
c
c
      subroutine JOHNSO(d,nd,nf,ind)
c     *****************
      integer nd,nf
      real d(nd,nd)
      logical ind

      integer pci0,pcj0,dci0,dcj0
      integer pc(40),dc(40),nc(40),sc(40)

      if((nf.lt.1).or.(nf.gt.3))nf=3

      n=nd

      do 1000 i=1,n
      pc(i)=i
      dc(i)=i
      nc(i)=i
      sc(i)=0
1000  continue

1500  if(n.eq.1)goto 20000
      i0=1
      j0=2
      i=nc(pc(i0))
      j=nc(pc(j0))
      if(j.gt.i)goto 2000
      x0=d(j,i)
```

```
        goto 3000
2000    x0=d(i,j)

3000    do 5000 k=2,n
        do 5000 l=1,k-1
        i=nc(pc(k))
        j=nc(pc(l))
        if(j.ge.i)goto 4000
        m=i
        i=j
        j=m
4000    x1=d(i,j)
        if(ind)goto 4500
        if(x0.ge.x1)goto 5000
        i0=l
        j0=k
        x0=x1
        goto 5000
4500    if(x0.le.x1)goto 5000
        i0=l
        j0=k
        x0=x1
5000    continue

        pci0=pc(i0)
        dci0=dc(i0)
        pcj0=pc(j0)
        dcj0=dc(j0)

        do 7000 i=i0,n-2
        if(i0.gt.n-2)goto 7000
        if(i.lt.(j0-1))goto 6000
        pc(i)=pc(i+2)
        dc(i)=dc(i+2)
        goto 7000
6000    pc(i)=pc(i+1)
        dc(i)=dc(i+1)
7000    continue

        p1=pci0
8000    if(p1.eq.0)goto 13000
        p2=pcj0
        i=nc(p1)
9000    if(p2.eq.0)goto 12000
        j=nc(p2)
        if(i.lt.j)goto 10000
        d(j,i)=x0
        goto 11000

10000   d(i,j)=x0
11000   p2=sc(p2)
        goto 9000
12000   p1=sc(p1)
        goto 8000
13000   i1=nc(pci0)
        i2=nc(pcj0)
        do 19000 i=1,n-2
        if(n.eq.2)goto 19000
        i3=nc(pc(i))
        if(i1.lt.i3)goto 14000
```

```
        x1=d(i3,i1)
        goto 15000
14000   x1=d(i1,i3)
15000   if(i2.lt.i3)goto 16000
        x2=d(i3,i2)
        goto 17000
16000   x2=d(i2,i3)
17000   call FORMUL(nf,x1,x2,res)
        if(i1.lt.i3)goto 18000
        d(i3,i1)=res
        goto 19000
18000   d(i1,i3)=res
19000   continue

        sc(dci0)=pcj0
        dc(n-1)=dcj0
        pc(n-1)=pci0

        n=n-1
        goto 1500

20000   return
        end
c
c
c
        integer function ULTRAM(tab,n,ind,pcname,axcode,pccode)
c       **********************
        real tab(n,n)
        integer n,pcname(2),axcode,pccode
        logical ind

        integer nci,gx(40),gy(39),dumi(2),axc,dir,nd,nda,pcc
        integer pcn(2)
        real xs,ys,abi(40),ori(39),yp(39),xg(39),xd(39),yg(39)
        real yd(39)
        common /areal0/nci,gx,gy,dumi,axc,dir,nd,nda,pcc,pcn
        common /areal1/xs,ys,abi,ori,yp,xg,xd,yg,yd

        integer tree(79,3),indiv(40),dum(2),grdx(40),grdy(39)
        integer dirg,result
        real ida(40),xpos(40),absi(40)
        real ypos(39),xposg(39),xposd(39),yposg(39),yposd(39)
        real ord(39)
        result=0

c       Verification des contraintes

        if((n.lt.2).or.(n.gt.40))goto 2000

        do 1000 i=1,n,1
        do 1000 j=1,i,1
        if(tab(i,j).ne.0.0)goto 3000
1000    continue

c       Trace du dendrogramme

        nida=n - 1
        ntree=2*n - 1
        xsize=100.
```

```
      ysize=60.
      dum(1)='      '
      dum(2)='      '
      dirg=2

      call DCLASS(tab,n,ind,tree,ntree,ida)
      call DEXPLO(tree,ntree,indiv)
      call DCALC(tree,ntree,ida,indiv,n,xpos,ypos,
     1          xposg,xposd,yposg,yposd,xsize,ysize)
      call DCONV(ida,indiv,n,xpos,ypos,absi,ord,nord,grdx,grdy)

      call jwindo(0.0,xsize,0.0,ysize)

      call DAXEH(absi,n,grdx,dum,axcode,dirg,2,xsize,ysize)
      call DAXEV(ord,nord,grdy,dum,axcode,1,5,xsize,ysize)
      call DRAW(nida,ypos,xposg,xposd,yposg,yposd,pccode)
      call TITRE(pcname,axcode,1,xsize,ysize)

c     Memorisation des donnees pour le trace dans un
c     metafichier

      xs=xsize
      ys=ysize
      dir=dirg
      nci=n
      nd=nord
      nda=nida
      axc=axcode
      pcc=pccode

      do 1500 i=1,2,1
      pcn(i)=pcname(i)
1500  dumi(i)=dum(i)

      do 1600 i=1,40,1
      gx(i)=grdx(i)
1600  abi(i)=absi(i)

      do 1700 i=1,39,1
      gy(i)=grdy(i)
      ori(i)=ord(i)
      yp(i)=ypos(i)
      xg(i)=xposg(i)
      xd(i)=xposd(i)
      yg(i)=yposg(i)
1700  yd(i)=yposd(i)

      goto 4000
2000  result=-1
      goto 4000
3000  result=-2
4000  ULTRAM=result
      return
      end
c
c
c
      subroutine PEND
c     ***************
```

```
      call jend

      return
      end
c
c
c
      subroutine TTYINI(device)
c     *****************
      integer device

      decode(5,100,device)it1,it2
100   format(a3,o2)
      open(unit=25,device=device,access='seqinout')
      call stty(it2)

      return
      end
c
c
c
      subroutine DAXEH(absi,nabs,grd,axname,axcode,dir,
     1                 nc,xsize,ysize)
c     ****************
      real absi(nabs),xsize,ysize
      integer grd(nabs),nabs,nc,axname(2),axcode,dir

      integer cl,st,lt,br

      call jopen

      call GADECO(axcode,cl,st,lt,br)

      call jcolor(cl)
      call jlstyl(st)
      call jlwide(lt)
      call jinten(br)

      call jsize(xsize/70,ysize/40)
      call jpath(1)
      call jjust(2,3)
      x0=xsize/20
      call jmove(- x0,0.0)
      call jdraw(xsize + x0,0.0)
      call jmove(xsize + x0/2,- x0)
      call jxtext(10,axname)

      call jpath(dir)

      do 1000 i=1,nabs,1
      x0=absi(i)
      call jmove(x0,0.0)
      call jdraw(x0,- ysize/100)
      call jmove(x0,- ysize/50)
      call jxtext(nc,grd(i))
1000  continue

      call jclose

      return
```

```
      end
c
c
c
      subroutine DAXEV(ord,nord,grd,axname,axcode,dir,
     1                 nc,xsize,ysize)
c     ***************
      real ord(nord),xsize,ysize
      integer grd(nord),nord,nc,axname(2),axcode,dir

      integer cl,st,lt,br

      call jopen

      call GADECO(axcode,cl,st,lt,br)

      call jcolor(cl)
      call jlstyl(st)
      call jlwide(lt)
      call jinten(br)

      call jsize(xsize/70,ysize/40)
      call jpath(1)
      call jjust(1,2)
      y0=ysize/12
      call jmove(0.0,- y0)
      call jdraw(0.0,ysize + y0)
      call jmove(- y0,ysize + y0/2)
      call jxtext(10,axname)

      call jpath(dir)
      call jjust(3,2)

      do 1000 i=1,nord,1
      y0=ord(i)
      call jmove(0.0,y0)
      call jdraw(- xsize/200,y0)
      call jmove(- xsize/100,y0)
      call jxtext(nc,grd(i))
1000  continue

      call jclose

      return
      end
c
c
c
      subroutine DCALC(tree,n,ida,indiv,nci,xpos,ypos,
     1                 xposg,xposd,yposg,yposd,xsize,ysize)
c     ***************
      integer n,nci,indiv(nci),tree(n,3)
      real xsize,ysize,ida(1)
      real xpos(1),ypos(1),xposg(1),xposd(1),yposg(1),yposd(1)

      real nux,nuy,nomc,nomp
      integer g,d

      nux=float(nci) + 1.
      nuy=float(nci)
```

```
      do 1000 i=1,nci,1
      il=i
      xpos(indiv(i))=float(il)*xsize/nux
1000  continue

      nomp=0.0
      do 2000 i=1,nci-1,1

      il=i
      nomc=ida(i)
      if(nomc.eq.nomp)goto 1500
      ypos(i)=float(il)*ysize/nuy
      nomp=nomc
      goto 2000
1500  ypos(i)=ypos(i-1)
2000  continue

      k=0

      do 3000 i=1,nci-1,1
      k=nci+i
      g=tree(k,1)
      d=tree(k,3)

      if(g.le.nci)goto 2100
      xposg(i)=xposg(g-nci)+(xposd(g-nci)-xposg(g-nci))/2.
      yposg(i)=ypos(g-nci)
      goto 2200
2100  xposg(i)=xpos(g)
      yposg(i)=0.

2200  if(d.le.nci)goto 2300
      xposd(i)=xposg(d-nci)+(xposd(d-nci)-xposg(d-nci))/2.
      yposd(i)=ypos(d-nci)
      goto 3000
2300  xposd(i)=xpos(d)
      yposd(i)=0.

3000  continue
      return
      end
c
c
c
      subroutine DCLASS(d,nd,ind,tree,nb,ynom)
c     *****************
      integer nd,nb,tree(nb,3)
      real d(nd,nd),ynom(1)
      logical ind

      integer td(40),ti(40),ta(40)
      integer tii0,tai0,taj0,tdi0,tdj0,free

      n=nd
      free=n+1
      m=1

      do 1000 i=1,n,1
      ti(i)=i
```

```
        ta(i)=i
        td(i)=i
1000    continue

        do 2000 i=1,n,1
        tree(i,1)=0
        tree(i,2)=i
        tree(i,3)=0
2000    continue

2500    if(n.eq.1)goto 11000

        k=ti(1)
        l=ti(2)
        if(k.lt.l)goto 3000
        x0=d(l,k)
        goto 4000
3000    x0=d(k,l)
4000    i0=l
        j0=2
        do 8000 j=2,n,1
        do 8000 i=1,j-1,1
        k=ti(i)
        l=ti(j)
        if(k.lt.l)goto 5000
        xl=d(l,k)
        goto 6000
5000    xl=d(k,l)
6000    if(ind)goto 7000
        if(xl.le.x0)goto 8000
        i0=i
        j0=j
        x0=xl
        goto 8000
7000    if(xl.ge.x0)goto 8000
        i0=i
        j0=j
        x0=xl
8000    continue

        ynom(m)=x0
        m=m+1

        tii0=ti(i0)
        tai0=ta(i0)
        taj0=ta(j0)
        tdi0=td(i0)
        tdj0=td(j0)

        do 10000 i=i0,n-2,1
        if(i0.gt.n-2)goto 10000
        if(i.lt.(j0-1))goto 9000
        ti(i)=ti(i+2)
        ta(i)=ta(i+2)
        td(i)=td(i+2)
        goto 10000
9000    ti(i)=ti(i+1)
        ta(i)=ta(i+1)
        td(i)=td(i+1)
10000   continue
```

```
      tree(free,1)=tai0
      tree(free,2)=free
      tree(free,3)=taj0
      tree(tdi0,3)=free
      ta(n-1)=free
      td(n-1)=tdj0
      ti(n-1)=tii0

      free=free+1
      n=n-1
      goto 2500

11000 continue

      return
      end
c
c
c
      subroutine DCONV(ida,indiv,n,xpos,ypos,absi,ord,nord,
     1                 grdx,grdy)
c     ****************
      integer n,nord,indiv(n),grdx(1),grdy(1)
      real ida(1),xpos(1),ypos(1),absi(1),ord(1)

      real nomp,nomc

      nomp=0.0
      nord=0

      do 1000 i=1,n-1,1
      nomc=ida(i)
      if(nomc.eq.nomp)goto 1000
      nomp=nomc
      nord=nord + 1
      ord(nord)=ypos(i)
      ENCODE(5,1,grdy(nord))ida(i)
1     format(f5.2)
1000  continue

      do 2000 i=1,n,1
      absi(i)=xpos(indiv(i))
      ENCODE(2,2,grdx(i))indiv(i)
2     format(i2)
2000  continue

      return
      end
c
c
c
      subroutine DEXPLO(tree,n,nom)
c     *****************
      integer n,tree(n,3),nom(1)

      integer x

      k=1
      x=tree(n,2)
```

```
500     if(tree(x,3).eq.0)goto 3000
1000    if(tree(x,1).eq.0)goto 2000

        x=tree(x,1)
        goto 1000
2000    nom(k)=tree(x,2)
        k=k+1
        x=tree(tree(x,3),3)
        goto 500
3000    nom(k)=tree(x,2)

        return
        end
c
c
c
        subroutine DRAW(nida,ypos,xposg,xposd,yposg,yposd,
        1                pccode)
c       ***************
        integer nida,pccode
        real ypos(1),xposg(1),xposd(1),yposg(1),yposd(1)

        integer cl,st,lt,br
        real dx(3),dy(3)

        call jopen

        call GADECO(pccode,cl,st,lt,br)

        call jcolor(cl)
        call jlstyl(st)
        call jlwide(lt)
        call jinten(br)

        do 1000 i=1,nida,1
        dx(1)=xposg(i)
        dx(2)=xposd(i)
        dx(3)=dx(2)
        dy(1)=ypos(i)
        dy(2)=dy(1)
        dy(3)=yposd(i)
        dx0=dx(1)
        dy0=yposg(i)
        call jmove(dx0,dy0)
        call jpoly(dx,dy,3)
1000    continue

        call jclose

        return
        end
c
c
c
        subroutine FORMUL(nf,a,b,res)
c       ******************
        integer nf
        real a,b,res
```

```
      res=0.0

      if(nf.eq.1)res=AMAX1(a,b)
      if(nf.eq.2)res=AMIN1(a,b)
      if(nf.eq.3)res=(a + b)/2.

      return
      end
c
c
c
      subroutine GADECO(code,cl,st,lt,br)
c     *****************
      integer code,cl,st,lt,br

      integer backg
      common /area5/backg

      itmp1=code
      cl=IFIX(itmp1/1000.)

      itmp2=MOD(itmp1,1000)
      st=IFIX(itmp2/100.)

      itmp3=MOD(itmp2,100)
      lt=IFIX(itmp3/10.)
      br=MOD(itmp3,10)

      if((cl.lt.1).or.(cl.gt.8))goto 4000
1000  if((st.lt.1).or.(st.gt.4))st=4
2000  if((lt.lt.1).or.(lt.gt.3))goto 5000
      lt=(lt - 1)*16383
3000  if((br.lt.1).or.(br.gt.3))goto 6000
      br=(br - 1)*16383
      goto 7000

4000  do 4500 i=1,8,1
      if(i.ne.backg)goto 4600
4500  continue
4600  cl=i
      goto 1000
5000  lt=16383
      goto 3000
6000  br=16383

7000  return
      end
c
c
c
      subroutine HCALC(rep,lim,n,ord)
c     ***************
      integer n,rep(n)
      real lim(1),ord(n)

      real h(40)

      hmax=0.0

      do 1000 i=1,n,1
```

```
      etend=lim(i+1) - lim(i)
      h(i)=float(rep(i)) / etend
      if(h(i).le.hmax)goto 1000
      hmax=h(i)
1000  continue

      do 2000 i=1,n,1
      ord(i)=h(i)/hmax
2000  continue

      return
      end
c
c
c
      subroutine HCONV(lim,nlim,xsize,absi,grd)
c     ****************
      real lim(nlim),absi(nlim),xsize
      integer nlim,grd(nlim)

      etend=lim(nlim) - lim(1)
      ratio=xsize/etend
      do 1000 i=1,nlim,1
      absi(i)=(lim(i) - lim(1))*ratio
      ENCODE(5,1,grd(i))lim(i)
1     format(f5.1)
1000  continue

      return
      end
c
c
c
      subroutine HDECO(code,color,shade)
c     ***********************
      integer code,color,shade

      integer backg
      common /area5/backg

      itmp1=code
      color=IFIX(itmp1/10.)
      shade=MOD(itmp1,10)

      if((color.lt.1).or.(color.gt.8))goto 1000
500   if((shade.lt.1).or.(shade.gt.7))shade=1
      if(shade.eq.7)shade=shade + 40
      goto 4000
1000  do 2000 i=1,8,1
      if(i.ne.backg)goto 3000
2000  continue
3000  color=i
      goto 500
4000  return
      end
c
c
c
      subroutine HDRAW(n,lim,h,ind,ecode,icode)
c     ****************
```

```
      integer n,ind(n),ecode(n),icode(n)
      real lim(1),h(n)

      integer ecl,est,elt,ebr
      real x(4),y(4)

      call jopen

      do 3000 i=1,n,1

      call GADECO(ecode(i),ecl,est,elt,ebr)

      call jcolor(ecl)
      call jlstyl(est)
      call jlwide(elt)
      call jinten(ebr)

      if(ind(i).eq.1)goto 1000
      call jpintr(0)
      goto 2000
1000  call jpintr(1)

      call HDECO(icode(i),icl,ist)
      call jpidex(icl,ist)

2000  x(1)=lim(i)
      y(1)=0.0
      x(2)=x(1)
      y(2)=h(i)
      x(3)=lim(i + 1)
      y(3)=y(2)
      x(4)=x(3)
      y(4)=y(1)
      call jpolgn(x,y,4)
3000  continue

      call jclose

      return
      end
c
c
c
      subroutine HPROJ(rep,lim,h,n,xsize,ysize)
c     ****************
      integer rep(n),n
      real lim(1),h(n),xsize,ysize

      integer string

      call jopen

      call jpath(1)
      call jjust(3,2)
      call jlstyl(2)
      call jsize(xsize/70,ysize/30)

      do 1000 i=1,n,1
      call jmove(lim(i),h(i))
      call jdraw(lim(1),h(i))
```

```
1000    continue
        do 2000 i=1,n,1
        call jmove(lim(1),h(i))
        ENCODE(5,1,string)rep(i)
1       format(i5)
        call jxtext(5,string)
2000    continue

        call jclose

        return
        end
c
c
c
        subroutine TITRE(pcname,tcode,dir,xsize,ysize)
c       ***************
        integer pcname(2),tcode,dir
        real xsize,ysize

        integer cl,dum

        call jopen

        call GADECO(tcode,cl,dum,dum,dum)

        call jcolor(cl)

        call jsize(xsize/70,ysize/40)
        call jpath(dir)
        call jjust(2,1)
        call jmove(xsize/2,ysize)
        call jxtext(10,pcname)

        call jclose

        return
        end
c
c
c
```

FACULTES UNIVERSITAIRES NOTRE-DAME DE LA PAIX (NAMUR)

INSTITUT D'INFORMATIQUE

REALISATION D'UN PROGICIEL GRAPHIQUE

ANNEXE A

MANUEL D'UTILISATION

Ph. Mataigne

Y. Perozzo

Année académique 1981 - 1982

FACULTES UNIVERSITAIRES NOTRE-DAME DE LA PAIX (NAMUR)

INSTITUT D'INFORMATIQUE

REALISATION D'UN PROGICIEL GRAPHIQUE

ANNEXE A

MANUEL D'UTILISATION

Ph. Mataigne

Y. Perozzo

Année académique 1981 - 1982

ANNEXE A : MANUEL UTILISATEUR DES PROCEDURES.

Ce document est le manuel d'utilisation des procédures constituant le progiciel graphique. Il comprend deux chapitres :

- Le premier chapitre correspond à la partie du travail écrite en Pascal. On y trouve les spécifications des procédures du générateur Pascal de métafichier, ainsi que les procédures relatives au tracé de fonctions d'une variable, d'histogrammes et de nuages de points dans le cadre d'une analyse en composantes principales.

- Le second chapitre correspond à la partie du travail écrite en Fortran. Ce chapitre reprend les spécifications des procédures relatives à la représentation et à la manipulation de graphes, au tracé d'histogrammes et à la représentation de classifications hiérarchiques.

Pour chaque procédure, on retrouve la structure suivante :

BUT : définition succinte de l'effet de la procédure.

ACCES : liste des éléments permettant l'accès à la procédure.

PARAMETRES : liste des paramètres de la procédure avec, éventuellement :

- les préconditions relatives aux paramètres d'entrée,

- les postconditions relatives aux paramètres de sortie.

DISCUSSION : considérations générales sur la procédure.

CONDITION D'ERREUR : liste des erreurs détectées lors de l'exécution de la procédure.

PROGRAMMATION : considérations relatives à certains points de programmation.

CHAPTER 1
PARTIE PASCAL.

1.1. INTRODUCTION.

Cette partie est divisée en deux :

- les procédures du générateur de métafichier,
- les procédures d'application.

1.2. LISTE ALPHABETIQUE DES PROCEDURES DU GENERATEUR PASCAL DE METAFICHIER.

1.2.1. PRELIMINAIRES.

1.2.1.1. Les classes de procédures.

Pour faciliter la manipulation des procédures par un programmeur, voici la liste de ces dernières par classes. Cette découpe est basée sur [DI][1], annexe VI, pages 2, 3 et 4.

- procédures accessibles aux programmeurs.
  - commandes de positionnement.
    - AMOVE (Absolute Move)
    - ADRAW (Absolute Draw)
  - commandes de non positionnement.
    - classe 0 : contrôle.
      - BPICT (Begin Picture)
      - EPICT (End Picture)
      - INITMF (Init Metafile)
      - ENDMF (End Metafile)
    - classe 1 : modes et marqueurs.
      - SETMS (Set Marker Symbol)
      - OUTSM (Output Marker Symbol)
      - BPOLYL (Begin Polyline)
      - EPOLYL (End Polyline)
      - BPOLYM (Begin Polymarker)
      - EPOLYM (End Polymarker)
      - BPOLYG (Begin Polygon)
      - EPOLYG (End Polygon)

---

[1] Les [ ] renvoient à la bibliographie du mémoire

- classe 2 : texte.

  - OUTSTR (Output String)
  - SETTFT (Set Text Font)
  - SETCSZ (Set Character Size)
  - SETCG (Set Character Gap)
  - SETTP (Set Text Precision)
  - SETTJ (Set Text Justification)
  - SETCBS (Set Character Base)

- classe 3 : attributs.

  - SETCOL (Set Color)
  - SETST (Set Style)
  - SETLW (Set Line Width)
  - SETPEN (Set Pen)
  - POLEST (Polygon Edge Style)
  - POLIST (Polygon Interior Style)
  - POLFC (Polygon Fill Color)
  - POLFI (Polygon Fill Intensity)

- procédures non accessibles aux programmeurs.

  (pour mémoire)

1.2.1.2. Programmation : Le contrôle des erreurs.

Afin d'éviter des séries de tests trop importantes et pour faciliter la compréhension des procédures du générateur, il a été décidé de ne prévoir des conditions et des messages d'erreurs que dans les procédures qui seront appelées explicitement dans le programme d'application de l'utilisateur. Comme la plupart des procédures du générateur sont destinées à faire partie d'autres procédures de plus haut niveau, les préconditions exprimées éventuellement dans la rubrique PARAMETRES de chaque procédure ne sont pas vérifiées ici, et on considère donc que ce sont les procédures de haut niveau qui garantissent la valeur correcte des paramètres. Le non respect d'une précondition lors de l'appel à une procédure a des effets dépendant de l'interpréteur de métafichier.

1.2.1.3. Programmation : Le format du métafichier.

Les métafichiers créés par le générateur doivent, pour être accessibles à l'interpréteur, respecter un certain format. Le lecteur pourra, s'il le désire, avoir tous les détails concernant ce dernier en consultant [DI]. D'autre part, ce format a fait l'objet d'une standardisation, que l'on pourra trouver en [PSG].

1.2.1.4. Programmation : La structure de données.

```
type metafile=file of integer;
     mfdesc=record
              word:mot;
```

```
                    Mot de 36 bits dans lequel est rangé
                    la commande (ou partie de commande) à
                    écrire dans le métafichier.
                nbre:integer;
                    Entier indiquant à tout moment combien
                    de groupes de 16 bits ont été écrits
                    dans le métafichier.
                pos:boolean
                    Booléen indiquant, s'il vaut true, que
                    le tampon WORD peut être utilisé dans
                    son entièreté, alors que s'il vaut
                    false, seule la partie droite de WORD
                    peut encore servir à stocker une
                    (partie de) commande.
            end;
```

1.2.2. LES PROCEDURES.

****

ADRAW

BUT:

Mouvement plume basse de la position courante à un point spécifié.

ACCES:

```
ADRAW(var m:metafile;x,y:integer)
```

PARAMETRES:

m : le métafichier dans lequel on écrit,
x : l'abscisse du point destination
précond. : 0 <= x <= 32767.
y : l'ordonnée du point destination
précond. : 0 <= y <= 32767.

****

AMOVE

BUT:

Mouvement plume haute de la position courante à un point spécifié.

ACCES:

```
AMOVE(var m:metafile;x,y:integer)
```

PARAMETRES:

m : le métafichier dans lequel on écrit,
x : l'abscisse du point destination
précond. : 0 < = x < = 32767.
y : l'ordonnée du point destination
précond. : 0 < = y < = 32767.

****

BMETA

BUT:

Initialisation du métafichier.

ACCES:

BMETA(var m:metafile)

PARAMETRES:

m : le métafichier dans lequel on écrit.

DISCUSSION:

Cette procédure fait partie de INITMF.

****

BPICT

BUT:

Initialisation d'un dessin.

ACCES:

BPICT(var m:metafile;i:integer)

PARAMETRES:

m : le métafichier dans lequel on écrit,
i : le numéro du dessin que l'on commence,
précond. : 0 < = i < = 32767.
Si ce n'est pas le cas, i est mis à la valeur par défaut 1.

DISCUSSION:

- Cette commande délimite le début d'un dessin.
- L'appel à BPICT doit au moins être précédé par un appel à INITMF.

ival est une variable globale qui permet de vérifier ce fait.

CONDITION D'ERREUR:

L'appel à BPICT n'est pas précédé par un appel à INITMF.
message : " l'appel à INITMF n'a pas été fait ".

PROGRAMMATION:

Cette commande doit obligatoirement se trouver dans le métafichier au début d'un groupe de 180 mots machine, d'où l'appel à REMPL au début de la procédure.

****

BPOLYG

BUT:

Initialisation du mode polygone.

ACCES:

BPOLYG(var m:metafile)

PARAMETRES:
---

m : le métafichier dans lequel on écrit.

DISCUSSION:

Lorsque l'on veut bénéficier des attributs graphiques relatifs aux polygones (style intérieur,couleur intérieure) une solution correcte est

- ouvrir le mode polygone par un appel à BPOLYG,
- tracer le polygone complet par des appels à AMOVE et ADRAW,
- fermer le mode polygone par un appel à EPOLYG.

L'expérimentation montre que l'interpréteur de métafichier n'exécute aucun ordre implicitement lorsqu'il est dans le mode polygone. (cf BPOLYM)

****

BPOLYL

BUT:

Initialisation du mode polyline.

ACCES:

BPOLYL(var m:metafile)

PARAMETRES:
---

m : le métafichier dans lequel on écrit.

DISCUSSION:

Lorsque l'on veut joindre une série de points par des segments de droite, soit on fait une boucle contenant des appels à ADRAW, soit on encadre cette même boucle avec les appels à BPOLYL et EPOLYL.

L'expérimentation montre en effet que l'interpréteur n'exécute aucun ordre qui n'est pas explicitement écrit dans le métafichier. (cf BPOLYM)

****

BPOLYM

BUT:

Initialisation du mode polymarker.

ACCES:

BPOLYM(var m:metafile)

PARAMETRES:

m : le métafichier dans lequel on écrit.

DISCUSSION:

Lorsque l'on désire repérer sur un dessin un ensemble de points en traçant un marqueur à la position de chaque point, soit on fait une boucle contenant des appels à AMOVE et OUTSM, soit on ouvre le mode polymarker par un appel à BPOLYM, on fait autant d'appels à AMOVE que nécessaire, et on ferme le mode par un appel à EPOLYM; Lors de la lecture du métafichier par l'interpréteur, celui-ci genère de lui-même les ordres OUTSM nécessaires pour obtenir ce qu'on désirait.

****

DEFVW

BUT:

Définition de l'espace d'affichage.

ACCES:

DEFVW(num:integer;var m:metafile)

PARAMETRES:

num : valeur définissant le ratio de l'image,
précond: 0 < = num < = 32767.
m : le métafichier dans lequel on écrit.

DISCUSSION:

Cette procédure fait partie de INITMF.

****

ENDMF

BUT:

Clôture du métafichier.

ACCES:

ENDMF(var m:metafile)

PARAMETRES:
---

m : le métafichier dans lequel on écrit.

DISCUSSION:

- Cette commande délimite la fin du métafichier.
- Cette procédure doit toujours être la dernière de la librairie à être appelée par l'utilisateur dans son programme d'application.
- L'appel à ENDMF doit au moins être précédé par un appel à INITMF, BPICT et EPICT.
  ival est une variable globale qui permet de vérifier ce fait.

CONDITION D'ERREUR:

L'appel à ENDMF n'est pas précédé par un appel à INITMF et/ou BPICT et/ou EPICT
message "un appel à INITMF ou/et à BPICT ou/et à EPICT n'a pas été fait" .

****

EMETA

BUT:

Terminaison du métafichier.

ACCES:

EMETA(var m:metafile)

PARAMETRES:
---

m : le métafichier dans lequel on écrit.

DISCUSSION:

Cette procédure fait partie de ENDMF.

****

EPICT

BUT:

Terminaison d'un dessin.

ACCES:

EPICT(var m:metafile)

PARAMETRES:
----

m : le métafichier dans lequel on écrit.

DISCUSSION:

- Cette commande délimite la fin d'un dessin.
- L'appel à EPICT doit au moins être précédé par un appel à INITMF et un à BPICT.
  ival est une variable globale qui permet de vérifier ce fait.

CONDITION D'ERREUR:

L'appel à EPICT n'est pas précédé par un appel à INITMF et/ou BPICT
message "l'appel à INITMF ou/et à BPICT n'a pas été fait" .

****

EPOLYG

BUT:

Terminaison du mode polygone.

ACCES:

EPOLYG(var m:metafile)

PARAMETRES:
----

m : le métafichier dans lequel on écrit.

****

EPOLYL

BUT:

Terminaison du mode polyline.

ACCES:

EPOLYL(var m:metafile)

PARAMETRES:
----

m : le métafichier dans lequel on écrit.

****

EPOLYM

BUT:

Terminaison du mode polymarker.

ACCES:

EPOLYM(var m:metafile)

PARAMETRES:

m : le métafichier dans lequel on écrit.

****

INITMF

BUT:

Constitution de l'enregistrement de tête (" header record ") du métafichier

ACCES:

INITMF(num:integer;var m:metafile)

PARAMETRES:

num : valeur définissant le ratio de l'image,
précond. : 0 < = num < = 32767.
m : le métafichier dans lequel on écrit.

DISCUSSION:

- Cette commande délimite le début du métafichier.
- Le ratio de l'image est le rapport entre la longueur de l'axe y et celle de l'axe x.
  La longueur de l'axe x est toujours 32767, et num est celle de y.
- Si num vaut 19660,le dessin est rectangulaire et aura un ratio de 0.6, cad maximisé pour un écran vidéo tel que celui du GIGI; Par contre s'il vaut 32767,le dessin sera carré, cad maximisé pour une table traçante telle que la Philips 8151.
- Par défaut, le dessin sera rectangulaire.
- Cette procédure doit toujours être la première de la librairie à être appelée par l'utilisateur. La variable globale ival est utilisée pour la vérification de cette contrainte.

****

INMETA

BUT:

Ecriture d'un mot de 36 bits dans un métafichier.

ACCES:

INMETA(var m:metafile)

PARAMETRES:
---

m : le métafichier dans lequel on écrit.

DISCUSSION:

Cette procédure est appelée chaque fois que le tampon WORD est rempli, et peut donc être écrit dans le métafichier.

PROGRAMMATION.

Comme on ne peut écrire dans le métafichier que des mots de 36 bits, et que avant l'appel seuls les 32 premiers bits du mot ont été positionnés,il faut donc mettre les 4 derniers à 0 avant de faire l'écriture.

****

MDO

BUT:

Ecriture d'un ordre de mouvement dans le métafichier.

ACCES:

MDO(var m:metafile;x,y,cl0:integer)

PARAMETRES:
---

m : le métafichier dans lequel on écrit,
x : l'abscisse du point,
précond: 0 < = x < = 32767
y : l'ordonnée du point,
précond: 0 < = y < = 32767
cl0 : la classe de la commande,
précond: 0 < = cl0 < = 1.

DISCUSSION:

Si cl0 = 0, on a un ordre de déplacement plume haute (AMOVE),
si cl0 = 1, on a un ordre de déplacement plume basse (ADRAW).

****

NOOP

BUT:

Ecriture dans le métafichier d'un mot de 36 bits qui sera ignoré lors de toute lecture par l'interpréteur de métafichier.

ACCES:

NOOP(var m:metafile)

PARAMETRES:

m : le métafichier dans lequel on écrit.

****

OUTSM

BUT:

Ecriture d'un marqueur à une position spécifiée sur le dessin .

ACCES:

OUTSM(var m:metafile;x,y:integer)

PARAMETRES:

m : le métafichier dans lequel on écrit,
x : l'abscisse de la position où le marqueur est écrit,
précond. : 0 < = x < = 32767.
y : l'ordonnée de la position où le marqueur est écrit,
précond. : 0 < = x < = 32767.

****

OUTSTR

BUT:

Ecriture d'une chaîne de caractères de longueur spécifiée sur le dessin, à la position courante.

ACCES:

OUTSTR(var m:metafile;nom:string10;lgr:integer)

PARAMETRES:

m : le métafichier dans lequel on écrit,
nom : la chaîne de caractères que l'on écrit,
lgr : la longueur de la chaîne,
précond. : 1 < = lgr < = 10.

PROGRAMMATION.

On découpe d'abord la chaîne de caractères en autant de groupes de quatre qu'on peut et on les écrit dans le métafichier, puis on écrit le groupe de deux qui restait.

****

PARNB0

BUT:

Ecriture dans le métafichier d'un ordre de non-mouvement sans paramètres.

ACCES:

PARNB0(var m:metafile;cl0,ind0:integer)

PARAMETRES:

m : le métafichier dans lequel on écrit,
cl0 : la classe de la commande,
précond : 0 <= cl0 <= 4
ind0 : l'index de la commande,
précond : 0 <= ind0 <= 10

****

PARNB1

BUT:

Ecriture dans le métafichier d'un ordre de non-mouvement avec un paramètre.

ACCES:

PARNB1(var m:metafile;cl0,ind0,i:integer)

PARAMETRES:

m : le métafichier dans lequel on écrit,
cl0 : la classe de la commande,
précond : 0 <= cl0 <= 4
ind0 : l'index de la commande,
précond : 0 <= ind0 <= 10
i : la valeur du paramètre,
précond : 0 <= i <= 32767

****

PARNB2

BUT:

Ecriture dans le métafichier d'un ordre de non-mouvement avec deux paramètres.

ACCES:

PARNB2(var m:metafile;cl0,ind0,w,h:integer)

PARAMETRES:
---

m : le métafichier dans lequel on écrit,
cl0 : la classe de la commande,
précond : 0 <= cl0 <= 4
ind0 : l'index de la commande,
précond : 0 <= ind0 <= 10
w : la valeur du premier paramètre,
précond : 0 <= w <= 32767
h : la valeur du deuxième paramètre,
précond : 0 <= h <= 32767

****

PARNB3

BUT:

Ecriture dans le métafichier d'un ordre de non-mouvement avec trois paramètres.

ACCES:

PARNB3(var m:metafile;cl0,ind0,x,y,z:integer)

PARAMETRES:
---

m : le métafichier dans lequel on écrit,
cl0 : la classe de la commande,
précond : 0 <= cl0 <= 4
ind0 : l'index de la commande,
précond : 0 <= ind0 <= 10
x : la valeur du premier paramètre,
précond : 0 <= x <= 32767
y : la valeur du deuxième paramètre,
précond : 0 <= y <= 32767
z : la valeur du troisième paramètre,
précond : 0 <= z <= 32767.

****

POLEST

BUT:

Fixation du style du bord du (des) polygone(s) du dessin.

ACCES:

POLEST(var m:metafile;i:integer)

PARAMETRES:
---

m : le métafichier dans lequel on écrit,
i : le style désiré,
précond. : 0 <= i <= 32767.

DISCUSSION:

Si i=0, le bord respectera les attributs de style et de couleur définis par le dernier appel aux procédures SETST et SETCOL.

****

POLIST

BUT:

Fixation du type de l'intérieur du (des) polygone(s) du dessin.

ACCES:

POLIST(var m:metafile;i:integer)

PARAMETRES:
---

m : le métafichier dans lequel on écrit,
i : le type désiré,
précond. : 0 <= i <= 32767.

DISCUSSION:

Deux types d'intérieurs pour les polygones sont définis, à savoir Si i=0, l'intérieur sera vide,
i=1, l'intérieur sera non vide, et respectera les attributs fixés par POLFC etPOLFI.
Les autres valeurs du paramètre ont un résultat dépendant du contrôleur d'appareil.

****

POLFC

BUT:

Fixation de la couleur de l'intérieur du (des) polygone(s).

ACCES:

POLFC(var m:metafile;i:integer)

PARAMETRES:
---

m : le métafichier dans lequel on écrit,
i : la couleur désirée,
précond. : 0 <= i <= 32767.

DISCUSSION:

Sept couleurs ont été définies , à savoir :
1 : rouge

2 : vert
3 : jaune
4 : bleu
5 : magenta
6 : cyan
7 : blanc.

Les autres valeurs du paramètre ont un résultat dépendant du contrôleur d'appareil.

****

POLFI

BUT:

Fixation du type de remplissage du (des) polygone(s). (" hachurage ")

ACCES:

POLFI(var m:metafile;i:integer)

PARAMETRES:
----

m : le métafichier dans lequel on écrit,
i : le type désiré,
précond. : 0 <= i <= 32767.

DISCUSSION:

Sept types de hachurage ont été définis, à savoir
1 : bandes verticales,
2 : bandes horizontales,
3 : bandes obliques vers la droite,
4 : bandes obliques vers la gauche,
5 : 1 + 2 ,
6 : 3 + 4 ,
7 : bandes verticales (différent de 1).

Les autres valeurs du paramètre ont un résultat dépendant du contrôleur d'appareil.

****

REMPL

BUT:

Termine le remplissage d'un tampon de 180 mots machine avec des ordres ' no-operation '.

ACCES:

REMPL(var m:metafile)

PARAMETRES:
----

m : le métafichier dans lequel on écrit.

DISCUSSION:

Le métafichier est composé de groupes de commandes de taille fixe: 180 mots machine. Un groupe ne comportant pas assez de commandes pour atteindre cette taille doit donc être complété.

PROGRAMMATION.

La valeur de l'indice k est toujours = à 360 car dans EMETA et dans EPICT, qui sont les deux seules procédures qui peuvent précéder REMPL, j'appelle VERIF, ce qui me garantit que le nombre de mots machine occupés ne dépasse pas 360.

****

SETCBS

BUT:

Fixation de la direction de la base des caractères du texte.

ACCES:

SETCBS(var m:metafile;xcomp,ycomp,zcomp:integer)

PARAMETRES:
---

m : le métafichier dans lequel on écrit,
xcomp,ycomp,zcomp : des valeurs telles que
précond. : 0 <= valeur <= 32767.

DISCUSSION:

Cet attribut de texte n'étant respecté que par des caractères "logiciel", dont on ne dispose pas, cette procédure est sans effet actuellement. (cf SETTP).

****

SETCG

BUT:

Fixation de l'espacement inter-caractères du texte.

ACCES:

SETCG(var m:metafile;ratioh,ratiov:integer)

PARAMETRES:
---

m : le métafichier dans lequel on écrit,
ratioh : le ratio horizontal pour l'espacement,
précond. : 0 <= ratioh <= 32767,
ratiov : le ratio vertical pour l'espacement,

précond. : 0 <= ratiov <= 32767.

DISCUSSION:

Un écartement normal est obtenu avec des paramètres égaux tous les deux à 32767. Les autres valeurs du paramètre ont un effet dépendant du contrôleur d'appareil.

****

SETCOL

BUT:

Fixation de la couleur des traits du dessin.

ACCES:

SETCOL(var m:metafile;i:integer)

PARAMETRES:
---

m : le métafichier dans lequel on écrit,
i : le numéro de la couleur désirée,
précond. : 0 <= i <= 32767.

DISCUSSION:

Sept couleurs numérotées de 1 à 7 ont été définies, à savoir :

sur le GIGI	sur la table
1 : rouge	1 : rouge
2 : vert	2 : vert
3 : jaune	3 : noir
4 : bleu	4 : bleu
5 : magenta	5 : rouge
6 : cyan	6 : vert
7 : blanc.	7 : noir

Les autres valeurs du paramètre ont un résultat dépendant du contrôleur d'appareil.

****

SETCSZ

BUT:

Fixation de la taille des caractères du texte.

ACCES:

SETCSZ(var m:metafile;w,h:integer)

PARAMETRES:
---

m : le métafichier dans lequel on écrit,
w : la largeur des caractères,
précond. : 0 <= w <= 32767,

h : la hauteur des caractères,
précond. : 0 <= h <= 32767.

DISCUSSION:

Le respect de la taille de caractère spécifiée dépend du contrôleur de l'appareil sur lequel le dessin est affiché. Pour la Philips 8151, la variation est continue, alors que pour le GIGI on dispose de 16 tailles différentes prédéfinies et c'est le contrôleur d'appareil qui choisit la taille parmi ces 16 qui est la plus proche de la taille désirée.

****

SETLW

BUT:

Fixation de la largeur des traits du dessin.

ACCES:

SETLW(var m:metafile;i:integer)

PARAMETRES:
---

m : le métafichier dans lequel on écrit,
i : la largeur désirée,
précond. : 0 <= i <= 32767.

****

SETMS

BUT:

Fixation du type des marqueurs du dessin.

ACCES:

SETMS(var m:metafile;ms:integer)

PARAMETRES:
---

m : le métafichier dans lequel on écrit,
ms : le type du marqueur,
précond. : 0 <= ms <= 32767.

DISCUSSION:

Cinq marqueurs différents numérotés de 1 à 5 sont définis, à savoir :
Si ms=1, le marqueur est un " . "
ms=2, le marqueur est un " + "
ms=3, le marqueur est un " * "
ms=4, le marqueur est un " O "
ms=5, le marqueur est un " X " .

Les autres valeurs du paramètre ont un résultat dépendant du contrôleur d'appareil.

****

SETPEN

Procédure non testée.

****

SETST

BUT:

Fixation du style des traits du dessin.

ACCES:

SETST(var m:metafile;i:integer)

PARAMETRES:

m : le métafichier dans lequel on écrit,
i : l'indice du style désiré,
précond. : 0 <= i <= 32767.

DISCUSSION:

Quatre styles de ligne différents numérotés de 1 à 4 sont disponibles :
pour le GIGI : 1 : tirets (-----)
2 : trait d'axe (_._._._)
3 : pointillé (.....)
4 : solide.
pour la table traçante : 1 : solide
2 : pointillé
3 : tirets
4 : solide .

****

SETTFT

BUT:

Fixation du type des caractères du texte du dessin.

ACCES:

SETTFT(var m:metafile;font:integer)

PARAMETRES:

m : le métafichier dans lequel on écrit,
font : le type des caractères,
précond. : 0 <= font <=32767 .

DISCUSSION:

Deux types de caractères numérotés 1 et 2 sont définis, à savoir :
Si font = 1, l'écriture se fera en lettres bloc (" block letters "),
= 2, l'écriture se fera en lettres italiques.
Les autres valeurs du paramètre ont un résultat dépendant du contrôleur d'appareil.

****

SETTJ

BUT:

Fixation de la justification horizontale et verticale du texte.

ACCES:

SETTJ(var m:metafile;hj,vj:integer)

PARAMETRES:
---

m : le métafichier dans lequel on écrit,
hj : la justification horizontale du texte,
précond. : 0 <= hj <= 3,
vj : la justification verticale du texte,
précond. : 0 <= hj <= 3.

DISCUSSION:

Comme on ne dispose que des caractères "matériels",et que pour ce type de caractères la justification n'est (en principe) pas respectée, cette procédure est donc inutilisée pour le moment.

****

SETTP

BUT:

Fixation de la précision du texte.

ACCES:

SETTP(var m:metafile;tp:integer)

PARAMETRES:
---

m : le métafichier dans lequel on écrit,
tp : la précision du texte,
précond. : 1 <= tp <= 3.

DISCUSSION:

Il y a trois précisions de texte prévues :
1 : string precision.
2 : character precision,
3 : stroke precision,
La première précision correspond aux caractères "matériels", alors que les deux dernières correspondent à des caractères générés par logiciel. Ceux-ci sont beaucoup mieux dessinés. Comme cette génération de caractères par logiciel n'est pas prise en charge par l'interpréteur de métafichier, et n'est pas, par manque de temps, realisée dans ce générateur de métafichier, on ne dispose pour le moment que des caractères matériels. Cela rend l'appel à des procédures comme SETTJ ou SETCBS inutile, étant donné que les attributs fixés par ces procédures ne sont respectés que par des caractères "logiciels".

****

SET2D

BUT:

Définition du nombre de dimensions du dessin.

ACCES:

SET2D(var m:metafile)

PARAMETRES:

m : le métafichier dans lequel on écrit.

DISCUSSION:

Cette procédure fait partie de INITMF.

****

VERIF

BUT:

Vérification qu'une commande tient entièrement dans le tampon courant de 180 mots.

ACCES:

VERIF(var m:metafile;pp:integer)

PARAMETRES:

m : le métafichier dans lequel on écrit,
pp : nombre de groupes de 16 bits occupés par la commande.

DISCUSSION:

Une commande ne peut chevaucher deux tampons de 180 mots. Cette procédure vérifie si le tampon courant peut contenir entièrement la commande.Si c'est le cas,on ne fait rien, sinon on complète le tampon courant avec des ordres no-operation et on recommence un nouveau tampon.

PROGRAMMATION.

Pour ce faire il suffit d'additionner à la place déjà occupée dans le tampon de 180 mots machine la place que prendrait la commande courante si elle y etait écrite, et de vérifier que la valeur obtenue ne dépasse pas les 180 mots permis.

****

1.3. LISTE DES PROCEDURES D'APPLICATION.

1.3.1. PRELIMINAIRES.

1.3.1.1. Programmation : La découpe en niveaux.

Cette librairie contient deux types de procédures.

Premièrement les procédures d'application proprement dites, qui sont destinées à être appelées dans un programme d'application écrit par l'utilisateur, et qui sont décrites dans le mémoire au chapitre 3. Ce sont les procédures ACP, FCT et HISTO, qui apparaissent en tête de la liste.

Deuxièmement, l'ensemble des sous-procédures classées par ordre alphabé tique.

1.3.1.2. Programmation : Le contrôle des erreurs.

En ce qui concerne les procédures ACP, FCT et HISTO, le contrôle se fait au début de chacune d'elles par des procédures specialisées.

Pour les autres, c'est la séquence de leurs appels dans ACP, FCT et HISTO qui garantit la correspondance deux à deux entre les postconditions de la procédure qui précède et les préconditions de celle qui suit.

1.3.1.3. Programmation : La structure de données.

```
type rtabl = array [1..500] of real;
     rtab2 = array [1..50] of real;
     ndctl = array [1..500] of integer;
     ndct2 = array [1..50] of integer;
     string10 = packed array [1..10] of char;
     achar = array [1..50] of string10;
     arw = record
               arwx : rtab2;
               arwy : rtab2;
           end;
     andc = record
                andcx : ndct2;
                andcy : ndct2;
            end;
     frw = record
               frwx : rtabl;
               frwy : rtabl;
           end;
     fndc = record
                fndcx : ndctl;
                fndcy : ndctl;
            end;
     zone = packed record
```

Remarque : dans les tableaux suivants, une valeur possible soulignée indique la valeur par défaut de la variable.

nom	type	signification	valeurs possibles
col :	0..777b	couleur du bord	1 à 7 (cf acol)
lst :	0..777b	style des lignes	1 à 4 (cf ast)
itype:	0..77b	type de l'intérieur	0 : intérieur vide 1 : intérieur non vide
fc :	0..77b	couleur de l'intérieur	1 à 7 (cf acol)
ist :	0..77b	style de l'intérieur	1 : ||| 2 : ≡ 3 : ⫽ 4 : ⫽ 5 : # 6 : ⨯ 7 : ||||
dim :	integer	nbre de pts de la courbe contenus ds la zone	0 à 500
nom :	string10	nom	1 à 10 caractères

```
            end;
        zones = array[1..40] of zone;
        opt = record
```

nom	type	signification	valeurs possibles
cars :	integer	taille des caractères du dessin	1:petits 2:grands
acol :	integer	couleur du(des) axe(s) - sur le GIGI	1:rouge 2:vert 3:jaune 4:bleu 5:magenta 6:cyan 7:blanc
		- sur la table traçante	1:rouge 2:vert 3:noir 4:bleu 5:rouge 6:vert 7:noir
ast :	integer	style du(des) axe(s) - sur le GIGI	1:– – – 2:–·–· 3:····· 4:——
		- sur la table traçante	1:—— 2:····· 3:– – – 4:——
axtyp :	integer	type de l'axe X	1:numérique 2:alphanum.
adrw :	boolean	booléen (cf FCT)	

axnb :	integer	nbre de graduations de l'axe X	2 à 50
aynb :	integer	nbre de graduations de l'axe Y	2 à 50
axname:	string10	nom de l'axe X	1 à 10 car.
ayname:	string10	nom de l'axe Y	1 à 10 car.
fxconv:	boolean	booléen (cf FCT)	
fyconv:	boolean	inutilisé	
fpronb:	integer	nbre de pts de la courbe à projeter sur les axes	0 à 50
fnb :	integer	nbre de pts de la courbe	0 à 500
znb :	integer	nbre de zones définies pour cette courbe	1 à 40
z :	zones	ensemble des caractéristiques des zones définies pour la courbe	

end;

gval = record

nom	type	signification
grsize:	integer	longueur normalisée du trait indiquant la présence d'une graduation le long de l'axe.
not1 :	integer	distance normalisée axe - position du premier étage d'écriture des graduations.
not2 :	integer	distance normalisée axe - position du deuxième étage d'écriture des graduations.
not3 :	integer	distance normalisée axe - position du premier étage d'écriture des projections.
not4 :	integer	distance normalisée axe - position du deuxième étage d'écriture des projections.
anam :	integer	distance normalisée axe - position d'écriture du nom de l'axe.
dmax :	integer	distance normalisée minimale axe - bord du dessin
nbpt :	integer	nombre maximum de graduations ou de projections le long d'un axe (dépend de la taille de caractères choisie)

end;

metafile=file of integer;

1.3.1.4. Notations.

Dorénavant, nous appellerons valeur "initiale" une valeur exprimée dans le système de coordonnées du monde réel (ce dernier est fixé par l'utilisateur dans son programme d'application) .

De même nous appellerons valeur "normalisée" une valeur exprimée dans le système de coordonnées normalisées des appareils (dans ce système, toutes les coordonnées sont comprises entre 0 et 32767).

1.3.2. LES PROCEDURES.

1.3.2.1. Procédures d'application.

****

ACP

BUT:

Représentation par rapport à deux axes principaux, d'une part, de la positon de p variables d'une analyse en composantes principales dans l'espace R^n des individus et d'autre part, de la position de n individus de l'analyse dans l'espace R^p des variables.

ACCES:

```
ACP(var1,var2,ind1,ind2:rtab2;vnb,indnb:integer;var
    o:opt;ccol,vcol,icol:integer;var mf:metafile)
```

PARAMETRES:

var1,var2 : tableaux des valeurs des corrélations variables-axes principaux de l'analyse dans R^n

précond. : toutes les corrélations sont comprises entre -1 et 1

ind1,ind2 : tableaux des valeurs des projections des individus sur deux axes principaux de l'analyse dans R^p

vnb,indnb : nombre de variables et d'individus dans le problème.

précond. : 1 <= vnb,indnb <= 50

o : enregistrement

Pour la signification des variables et les contraintes existantes sur leurs valeurs, cf la structure de données (1.3.1.3.).

Remarque : Seuls les champs suivants de l'enregistrement sont utilisés.

cars , acol , ast , axname , ayname.

ccol,vcol,icol : couleurs utilisées pour le dessin du cercle et la représentation des variables et des individus.

précond. : 1 = ccol,vcol,icol = 7

diagnostic : remise à la valeur par défaut (7).

mf : le métafichier dans lequel on écrit.

DISCUSSION:

Cette procédure est étudiée dans le mémoire au point

3.2.3

NOTE IMPORTANTE :
La procédure ACP ne peut pas apparaître seule dans un programme d'application. Il y a une séquence d'appels de procédures à respecter.

SEQUENCE D'APPELS A RESPECTER:
---

1) Rappels.
- - - - - -

- Un métafichier est constitué d'un ou plusieurs dessins.

- Afin de pouvoir distinguer les différents dessins dans un même métafichier, il faut débuter chaque dessin par un appel à la procédure BPICT, et après avoir tracé le dessin, signaler que celui-ci est terminé par un appel à la procédure EPICT.

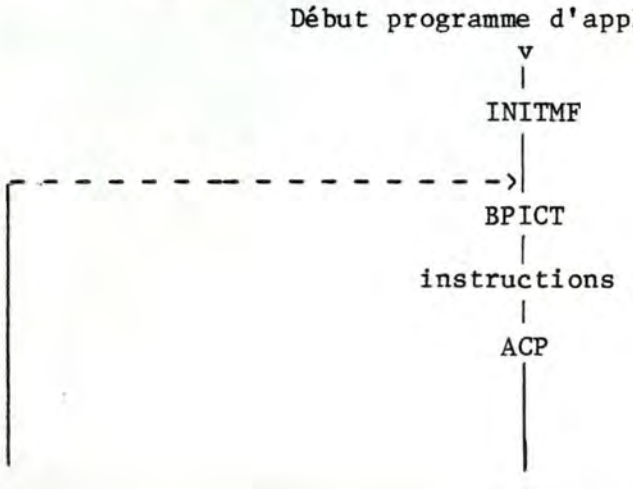
- De même, la procédure INITMF doit être appelée avant toute autre procédure de la librairie afin de délimiter le début du métafichier. Enfin, la procédure ENDMF doit être appelée après toutes les autres afin de signaler la fin du métafichier.

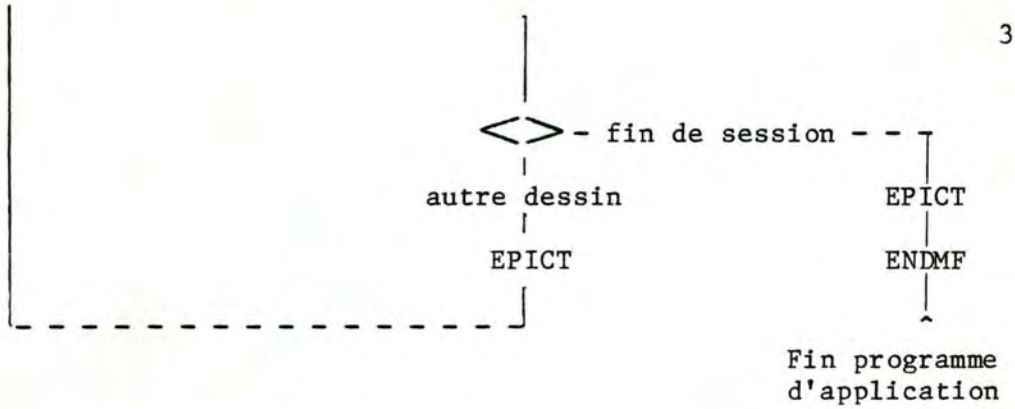
2) Contraintes.
- - - - - - - -

- Comme la procédure ACP a été prévue pour le tracé éventuel de plusieurs dessins dans un seul métafichier, il a donc été impossible d'inclure au début et à la fin de ACP les appels aux procédures citées ci-dessus.

- L'utilisateur doit donc lui-même, dans son programme d'application, placer l'appel à INITMF tout au début de son programme, après l'instruction Rewrite du métafichier par exemple. L'appel à ENDMF quant à lui peut être placé immédiatement avant le end terminal . De plus, au moins un appel à BPICT et un à EPICT doit apparaître de part et d'autre de l'appel à ACP.

- Nous pouvons représenter un exemple de programme d'application comme suit :





3) Spécifications.

Les procédures INITMF, ENDMF, BPICT et EPICT sont spécifiées dans l'annexe relative au générateur (Pascal) de métafichier.

4) Contrôle des contraintes.

Il existe au début de ACP un test permettant de vérifier si au moins un appel à INITMF et à BPICT a été fait.

CONDITION D'ERREUR:

1) Une des corrélations est négative.
   message : " une corrélation a une valeur non comprise entre -1 et 1 ".
2) La valeur de inb ou vnb n'est pas comprise entre 1 et 50.
   message : " Le nbre des var. ou des individus n'est pas compris entre 1 et 50 ".

PROGRAMMATION:

* variables, tableaux et enregistrements globaux à la librairie.
  - b : vaut true si une erreur a été détectée dans la chaîne des appels.
  - ival : vaut 2 si on a appelé INITMF et BPICT avant ACP.
  - p : enregistrement comprenant
    - a : tableau des valeurs normalisées des graduations des axes x et y.
    - c : tableau des valeurs normalisées des coordonnées des points du cercle unité.
    - tca : tableau des graduations des axes sous forme de chaînes de caractères.
    - tc : tableau des valeurs de 1 à 50 sous forme de chaînes de caractères.
    - px,py : valeurs normalisées des coordonnées en x et y du point d'intersection des deux axes.
    - gv : contient la position normalisée du texte autour des axes (cf la procédure CGVAL).

* variables, tableaux et enregistrements locaux à la procédure.
  - arwx,arwy : tableaux des valeurs initiales des graduations des axes x et y.

- min1,max1,min2,max2 : valeurs des minimums et des maximums des tableaux ind1 et ind2.
- av : contient les valeurs normalisées des coordonnées en x et y des variables du problème.
- ai : contient les valeurs normalisées des coordonnées en x et y des individus du problème.

* corps.
Le travail se fait en huit parties.
1) Test si les appels à INITMF et BPICT ont été faits.
2) Vérification des contraintes.
3) Construction des deux axes.
(ne se fait pas si on trace un deuxième dessin dans le même métafichier).
4) Dessin du système d'axes.
5) Construction du cercle.
(ne se fait pas si on trace un deuxième dessin dans le même métafichier).
6) Dessin du cercle.
7) Affichage de la position des variables par rapport aux deux axes.
8) Affichage de la position des individus par rapport aux deux axes.
a) recherche du maximum (en valeur absolue) des tableaux ind1 et ind2.
b) calcul des positions normalisées des individus dans un intervalle -max,max (valeurs initiales).
c) affichage proprement dit.
d) recherche de la position de la graduation de valeur initiale 1 sur l'axe horizontal.

****

FCT

BUT:

Dessin d'un système d'axes et d'une fonction d'une variable.

ACCES:

FCT(aw:arw;fw:frw;dchar:achar;p:ndct2;o:opt;var mf: metafile)

PARAMETRES:
----

aw : enregistrement comprenant
arwx : tableau des valeurs initiales des graduations de l'axe x
arwy : tableau des valeurs initiales des graduations de l'axe y
précond. : 1) format:
Une graduation ne peut dépasser cinq caractères (signe et point décimal valant chacun un caractère, le + est optionnel

et remplacé par un blanc).
diagnostic : un string vide remplace sur le dessin les graduations au dela de +- 9999.
2) ordre:
La suite des graduations doit être dans un ordre strictement croissant.
3) valeur:
Si les graduations passent de valeurs < 0 à des valeurs > 0, la valeur 0 doit être explicitement une graduation

fw : enregistrement comprenant
frwx : tableau des valeurs initiales des abscisses des points de la courbe.
frwy : tableau des valeurs initiales des ordonnées des points de la courbe.
précond. : valeur:
Toutes les coordonnées numériques doivent avoir une valeur comprise entre la première et la dernière graduation de l'axe correspondant.
diagnostic : Remplacement de la valeur erronée par la valeur de la graduation limite la plus proche.
message : " Attention, il y a eu clipping pour certaines val. de la fct !" .

dchar : tableau des graduations alphanumériques de l'axe x. Si celui-ci est de ce type, aw.arwx et fw.frwx sont ignorés par FCT.Sinon (si les deux axes sont numériques),c'est dchar qui est ignoré et aw.arwx et fw.frwx qui sont pris en compte.
précond. : format:
Une graduation ne peut dépasser cinq caractères.
diagnostic : seuls les cinq premiers caractères de chaque graduation sont écrits le long de l'axe.

p : tableau des indices (dans l'ensemble des points de la fonction) des points que l'on veut projeter.
précond. : valeur:
Tous les indices doivent être compris entre 1 et le nombre de points de la courbe.

o : enregistrement
Pour la signification des variables et les contraintes existantes sur leurs valeurs, cf la structure de données (1.3.1.3.).
Remarque :
---
o.adrw :
booléen qui lorsqu'il vaut TRUE entraîne le non dessin du système d'axes.
L'utilisation de cette variable permet

une économie de travail et de calculs lorsqu'on veut tracer plusieurs courbes sur un même dessin.
Il faut signaler ici qu'en Pascal, un booléen vaut FALSE lors de sa déclaration.

o.fxconv :
booléen qui lorsqu'il vaut TRUE entraîne une économie de calcul lorsqu'on trace consécutivement dans le même dessin deux courbes qui ne diffèrent que par les valeurs des ordonnées de leurs points.
Il faut signaler ici qu'en Pascal, un booléen vaut FALSE lors de sa déclaration.

o.fnb :
lorsque l'axe x est alphanumérique (axtyp=2), la variable o.fnb vaut toujours o.axnb.

o.z[i].dim :
les points de la courbe qui séparent deux zones doivent être comptés dans les deux zones

mf : le métafichier dans lequel on stocke le dessin.

DISCUSSION:

Cette procédure est étudiée dans le mémoire au point 3.2.1

NOTE IMPORTANTE :
La procédure FCT ne peut pas apparaître seule dans un programme d'application. Il y a une séquence d'appels de procédures à respecter.

SEQUENCE D'APPELS A RESPECTER:
---

1) Rappels.
- - - - - -

- Un métafichier est constitué d'un ou plusieurs dessins.

- Afin de pouvoir distinguer les différents dessins dans un même métafichier, il faut débuter chaque dessin par un appel à la procédure BPICT, et après avoir tracé le dessin, signaler que celui-ci est terminé par un appel à la procédure EPICT.

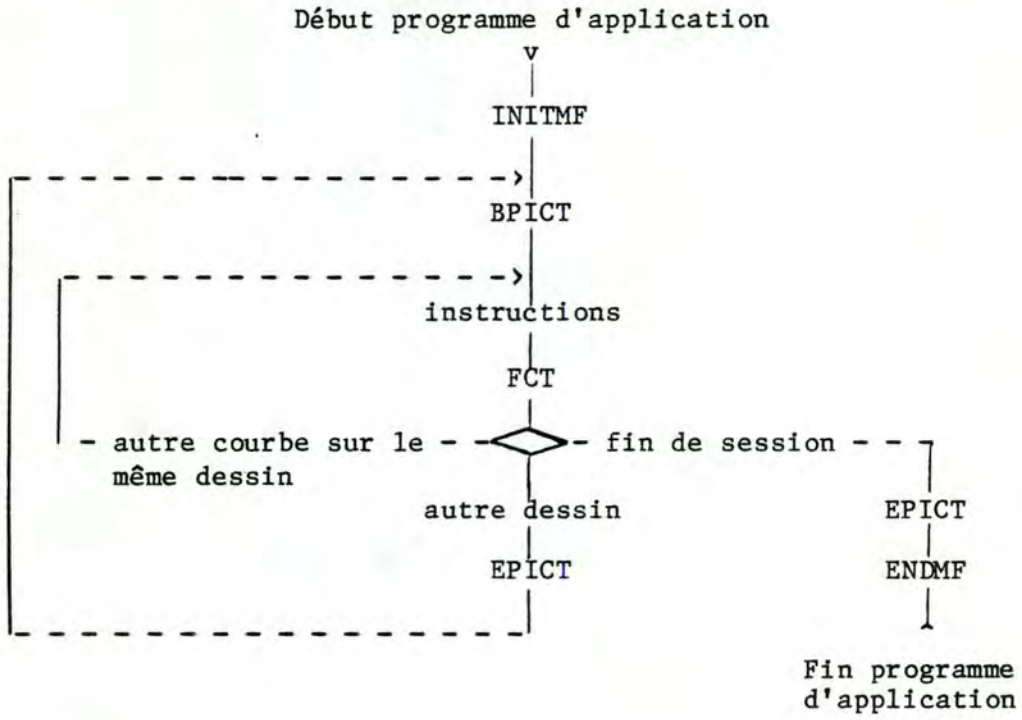
- De même, la procédure INITMF doit être appelée avant toute autre procédure de la librairie afin de délimiter le début du métafichier. Enfin, la procédure ENDMF doit être appelée après toutes les autres afin de signaler la fin du métafichier.

2) Contraintes.

- Comme la procédure FCT a été prévue pour le tracé éventuel de plusieurs courbes sur un même dessin, et de plusieurs dessins dans un seul métafichier, il a donc été impossible d'inclure au début et à la fin de FCT les appels aux procédures citées ci-dessus.

- L'utilisateur doit donc lui-même, dans son programme d'application, placer l'appel à INITMF tout au début de son programme, après l'instruction Rewrite du métafichier par exemple. L'appel à ENDMF quant à lui peut être placé immédiatement avant le end terminal . De plus, au moins un appel à BPICT et un à EPICT doit apparaître de part et d'autre de l'appel à FCT.

- Nous pouvons représenter un exemple de programme d'application comme suit :



3) Spécifications.

Les procédures INITMF, ENDMF, BPICT et EPICT sont spécifiées dans l'annexe relative au générateur (Pascal) de métafichier.

4) Contrôle des contraintes.

Il existe au début de FCT un test permettant de vérifier si au moins un appel à INITMF et à BPICT a été fait.

CONDITION D'ERREUR:

1) Les graduations de l'axe X ou Y ne sont pas dans un

ordre strictement croissant.
message : " Grad. de l'axe non strct ".

2) L'ensemble des graduations de l'axe X ou Y comprend des valeurs négatives et positives, mais ne comprend pas la valeur 0.
message : " La grad. 0 manque sur l'axe ".

3) La valeur de o.axnb ou o.aynb n'est pas comprise entre 2 et 50.
message : " valeur non permise pour une option ".

4) La valeur de o.fnb n'est pas comprise entre 2 et 500.
message : " valeur non permise pour une option ".

5) Une des valeurs de o.z[i].dim n'est pas comprise entre 2 et 500
message : " valeur non permise pour une option ".

6) L'indice d'un point à projeter de la courbe n'est pas compris entre 1 et o.fnb
message : " numéro de proj inexistant ".

PROGRAMMATION:
---

* variables, tableaux et enregistrements globaux à la librairie.
  - b : vaut true si une erreur a été détectée dans la chaîne des appels.
  - ival : vaut 2 si on a appelé INITMF et BPICT avant FCT.
  - rinv : vaut true s'il y a lieu d'inverser le dessin entier.
  - fd : contient les valeurs normalisées des coordonnées des points de la dernière courbe tracée. (utile si 2 courbes tracées consécutivement ont les mêmes abscisses).

* variables, tableaux et enregistrements locaux à la procédure.
  - ad : contient les valeurs normalisées des graduations des axes.
  - acharx,achary : tableaux des graduations des axes sous forme de chaînes de caractères.
  - posx,posy : valeur normalisée de b dans l'équation des axes : x(y) = b.
  - ga : contient la position normalisée du texte autour des axes (cf CGVAL).
  - prwx,prwy : tableaux des valeurs initiales des abscisses (ordonnées) des points à projeter de la courbe.
  - pndcx,pndcy : tableaux des valeurs normalisées des abscisses (ordonnées) des points à projeter de la courbe.

* corps.
  Le travail se fait en cinq parties.
  1) Test si les appels à INITMF et BPICT ont été faits.
  2) Vérification des contraintes.
     (ne se fait plus si on trace une deuxième courbe dans le même dessin).
  3) Dessin des axes.
     (ne se fait plus si on trace une deuxième courbe dans le même dessin).

4) Dessin de la courbe.
5) Dessin des projections.

****

HISTO

BUT:

Dessin d'un histogramme.

ACCES:

HISTO(aw:arw;o:opt;var mf:metafile)

PARAMETRES:

aw : enregistrement comprenant
arwx : tableau des valeurs initiales des limites de classes
précond. : 1) format:
Une graduation ne peut dépasser cinq caractères (signe et point décimal valant chacun un caractère, le + est optionnel et remplacé par un blanc).
diagnostic : un string vide remplace sur le dessin les graduations au dela de +- 9999.
2) ordre:
La suite des graduations doit être dans un ordre strictement croissant.
3) nombre:
Le nombre de limites de classes doit être compris entre 2 et 41.
arwy : tableau des répétitions
précond. : Toutes les répétitions doivent être positives.

o : enregistrement
Pour la signification des variables et les contraintes existantes sur leurs valeurs, cf la structure de données (1.3.1.3.).
Remarque : Seuls les champs suivants de l'enregistrement sont utilisés.

cars , acol , ast , axnb , axname , z[i].col
.lst
.itype
.fc
.ist
.nom

mf : le métafichier dans lequel on écrit.

DISCUSSION:

Cette procédure est étudiée dans le mémoire au point 3.2.2

NOTE IMPORTANTE :
La procédure HISTO ne peut pas apparaître seule dans un programme d'application. Il y a une séquence d'appels de procédures à respecter.

SEQUENCE D'APPELS A RESPECTER:
---

1) Rappels.
- - - - - -

- Un métafichier est constitué d'un ou plusieurs dessins.

- Afin de pouvoir distinguer les différents dessins dans un même métafichier, il faut débuter chaque dessin par un appel à la procédure BPICT, et après avoir tracé le dessin, signaler que celui-ci est terminé par un appel à la procédure EPICT.

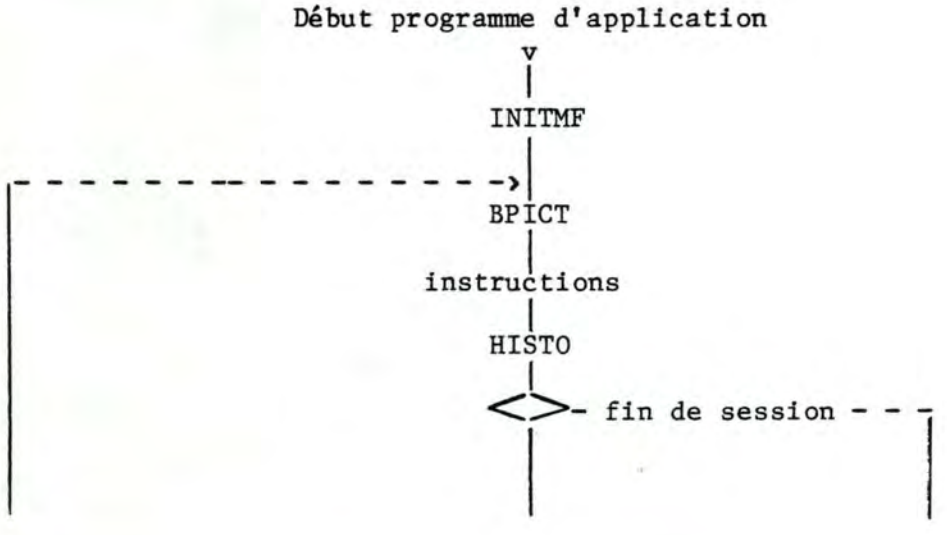
- De même, la procédure INITMF doit être appelée avant toute autre procédure de la librairie afin de délimiter le début du métafichier. Enfin, la procédure ENDMF doit être appelée après toutes les autres afin de signaler la fin du métafichier.

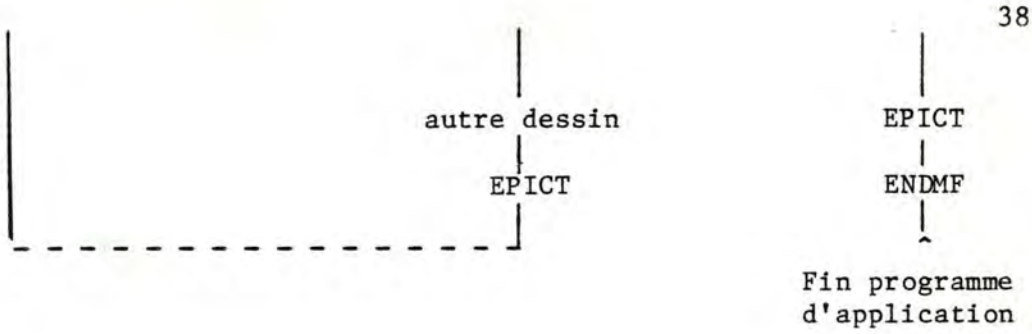
2) Contraintes.
- - - - - - - -

- Comme la procédure HISTO a été prévue pour le tracé éventuel de plusieurs dessins dans un seul métafichier, il a donc été impossible d'inclure au début et à la fin de HISTO les appels aux procédures citées ci-dessus.

- L'utilisateur doit donc lui-même, dans son programme d'application, placer l'appel à INITMF tout au début de son programme, après l'instruction Rewrite du métafichier par exemple. L'appel à ENDMF quant à lui peut être placé immédiatement avant le end terminal . De plus, au moins un appel à BPICT et un à EPICT doit apparaître de part et d'autre de l'appel à HISTO.

- Nous pouvons représenter un exemple de programme d'application comme suit :





3) Spécifications.

Les procédures INITMF, ENDMF, BPICT et EPICT sont spécifiées dans l'annexe relative au générateur (Pascal) de métafichier.

4) Contrôle des contraintes.

Il existe au début de HISTO un test permettant de vérifier si au moins un appel à INITMF et à BPICT a été fait.

CONDITION D'ERREUR.

1) Les limites de classe ne sont pas dans un ordre strictement croissant
   message : " Grad. de l'axe non strct > ".
2) La valeur de o.axnb n'est pas comprise entre 2 et 41.
   message : " valeur non permise pour une option ".
3) Une des répétitions est négative.
   message : " une répétition au moins est négative ! ".

PROGRAMMATION:

* variables, tableaux et enregistrements globaux à la librairie.
  - b : vaut true si une erreur a été détectée dans la chaîne des appels.
  - ival : vaut 2 si on a appelé INITMF et BPICT avant HISTO.

* variables, tableaux et enregistrements locaux à la procédure.
  - axe : tableau des valeurs normalisées des graduations de l'axe
  - hndc : tableau des valeurs normalisées des hauteurs des rectangles
  - ga : contient la position normalisée du texte autour des axes (cf CGVAL).
  - acharx : tableau des graduations de l'axe sous forme de chaînes de caractères.
  - posmin,posmax : position de min et de max dans l'ensemble des valeurs initiales des hauteurs.
  - min,max : valeurs minimale et maximale de l'ensemble des valeurs initiales des hauteurs.

* corps.
  Le travail se fait en cinq parties.
  1) Test si les appels à INITMF et BPICT ont été

faits.
2) Vérification des contraintes.
3) Dessin de l'axe.
4) Calcul de la hauteur et dessin des rectangles.
5) Ecriture des valeurs des répétitions.

****

1.3.2.2. Sous-procédures.

****

ADRAW2

BUT:

Dessin d'un système d'axes.

ACCES:

ADRAW2(ga:gval;acharx,achary:achar;var a:andc;posx,posy:
integer;o:opt;var mf:metafile)

PARAMETRES:

ga : enregistrement
Pour la signification des variables, cf la structure de données (1.3.1.3.).
précond. : 0 < = grsize,not1,not2,not3,not4,anam, dmax < = 32767
0 < = nbpt < = 50
acharx,achary : tableaux des valeurs initiales des graduations de l'axe X (respectivement Y)
a : enregistrement comprenant
andcx : tableau des valeurs normalisées des graduations de l'axe X
précond. : 0 < = andcx[i] < = 32767
andcy : tableau des valeurs normalisées des graduations de l'axe Y
précond. : 0 < = andcy[i] < = 32767
posx,posy : valeurs normalisées de b dans l'équation de l'axe X (respectivement Y) : y(resp. x)=b
o : enregistrement
pour la signification des variables et leurs valeurs admissibles, cf la structure de données (1.3.1.3.)
précond. : toutes les valeurs sont correctes.
mf : le métafichier dans lequel on écrit.

****

BOX

BUT:

Dessin d'un carré délimitant les bords du dessin.

ACCES:

BOX(var mf:metafile)

PARAMETRES:
----

mf : le métafichier dans lequel on écrit.

****

CCHAR

BUT:

Manipulation de tableaux.

ACCES:

```
CCHAR(tchar1:achar;ptnb:integer;var atab:rtab2;var
      tchar2:achar;var ftab:rtab1)
```

PARAMETRES:
----

tchar1 : tableau de caractères à recopier (voir tchar2)
ptnb : nombre d'éléments dans les tableaux à manipuler
précond. : 0 <= ptnb <= 50
atab : tableau résultat
postcond. : atab contient la suite 1,2,3,...,
ptnb.
tchar2 : tableau dans lequel tchar1 est recopié.
ftab : idem atab.

****

CERCLE

BUT:

Construction de l'ensemble des valeurs normalisées des coordonnées des points d'un cercle.

ACCES:

CERCLE(var fndcx,fndcy:ndctl;max:integer)

PARAMETRES:
----

fndcx,fndcy : tableaux des valeurs normalisées des abscisses (respectivement ordonnées) des points du cercle
postcond. : 0 <= fndcx[i],fndcy[i] <= 32767
max : nombre de valeurs dans fndcx (et fndcy)
précond. : 0 <= max <= 500.

DISCUSSION:

Le cercle a son centre en 0,0 et un rayon de 1 (valeurs initiales).

Rappel : équation d'un cercle de centre c et de rayon r :

$$x1^2 + x2^2 - 2c1x1 - 2c2x2 + c1^2 + c2^2 - r^2 = 0$$

PROGRAMMATION:

1) Calcul des valeurs initiales des coordonnées des points.
2) Conversion des valeurs initiales en valeurs normalisées.

****

CGVAL

BUT:

Fixation de la position du texte par rapport à l'axe.

ACCES:

CGVAL(size:integer;var g:gval)

PARAMETRES:

size : taille des caractères du dessin
g : enregistrement
Pour la signification des variables, cf la structure de données (1.3.1.3.).
postcond. : 0 <= grsize,not1,not2,not3,not4,anam, dmax <= 32767
0 <= nbpt <= 50

DISCUSSION:

Si la variable size = 1, on a choisi la petite taille de caractères sinon c'est la grande taille.

****

COMPL1

BUT:

Calcul du complément à 32767 de valeurs.

ACCES:

COMPL1(var tab:ndctl;ptnb:integer)

PARAMETRES:

tab : le vecteur des valeurs à complémenter
précond. : 0 <= tab[i] <= 32767

```
                postcond. : 0 < = tab[i] < = 32767
        ptnb : le nombre de valeurs dans tab
                précond. : 0 < = ptnb < = 500.

**********************************************************************

                                COMPL2
                                ------

BUT:
----
        Calcul du complément à 32767 de valeurs.

ACCES:
------
        COMPL2(var tab:ndctl;ptnb:integer)

PARAMETRES:
-----------
        tab : le vecteur des valeurs à complémenter
                précond. : 0 < = tab[i] < = 32767
                postcond. : 0 < = tab[i] < = 32767
        ptnb : le nombre de valeurs dans tab
                précond. : 0 < = ptnb < = 50.

**********************************************************************

                                CONSTR
                                ------

BUT:
----
        Remplissage d'un tableau avec les entiers de 1 à nb.

ACCES:
------
        CONSTR(var tab:rtab2;nb:integer)

PARAMETRES:
-----------
        tab : tableau résultat
                postcond. : tab contient la suite 1,2,3,...,nb
        nb : nombre limite de la suite
                précond. : 0 < = nb < = 50.

**********************************************************************

                                CONTRT
                                ------

BUT:
----
        Vérification que les points d'une courbe sont dans les
        limites de validité définies par les graduations des deux
        axes.

ACCES:
------
        CONTRT(a:arw;var f:frw;aptl,apt2,fptnb,typ:integer;x,y:
               boolean; var err:boolean)
```

PARAMETRES:
---

a : enregistrement comprenant
arwx : tableau des valeurs initiales des graduations de l'axe X
précond. : 0 < = arwx[i] < = 32767
arwy : tableau des valeurs initiales des graduations de l'axe Y
précond. : 0 < = arwy[i] < = 32767
f : enregistrement comprenant
frwx : tableau des valeurs initiales des abscisses des points de la courbe
frwy : tableau des valeurs initiales des ordonnées des points de la courbe
aptl,apt2 : nombre de valeurs dans a.arwx (respectivement a.arwy)
précond. : 0 < = aptl,apt2 < = 50
fptnb : nombre de valeurs dans f.frwx (et f.frwy)
précond. : 0 < = fptnb < = 500
typ : type de l'axe X
x,y : booléens
précond. : valent false s'il faut effectuer la vérification sur les abscisses (resp. ordonnées) des points de la courbe
err : booléen
postcond. : vaut true si au moins une des données a dû être rectifiée.

DISCUSSION:

Si la variable typ vaut 1 l'axe X est numérique sinon il est alphanumérique.
Lorsque la valeur d'une abscisse (ordonnée) est en-deçà de la première graduation de l'axe ou au-delà de la dernière, la variable est remise à la valeur de la première (ou respectivement de la dernière) graduation.

****

CONVAN

BUT:

Conversion des valeurs initiales des graduations d'un axe en valeurs normalisées et détermination de la valeur normalisée de l'intersection des deux axes.

ACCES:

```
CONVAN(rwtab:rtab2;dmax,aptnb:integer;var ndctab:ndct2;
       var pos:integer;var err:boolean)
```

PARAMETRES:
---

rwtab : tableau des valeurs initiales des graduations de l'axe.
précond. : les valeurs sont dans un ordre strictement croissant
dmax : distance normalisée minimum axe-bord du dessin

précond. : 0 < = dmax < = 32767
aptnb : nombre de valeurs dans rwtab
précond. : 0 < = aptnb < = 50
ndctab : tableau résultat des valeurs normalisées des graduations de l'axe
postcond. : 0 < = ndctab[i] < = 32767
pos : valeur normalisée de l'intersection de cet axe avec l'autre.
postcond. : 0 < = pos < = 32767
err : booléen
vaut true si un erreur a été détectée.

CONDITION D'ERREUR:

L'ensemble des valeurs initiales des graduations contient des valeurs négatives et positives et la valeur 0 n'appartient pas à cet ensemble.
message : " la grad. 0 manque sur l'axe ".

PROGRAMMATION:

Au maximum, un axe va de 1000 à 31000 (valeurs normalisées). Quatre cas peuvent se présenter :

1) Toutes les valeurs initiales sont strictement négatives = le dessin sera compris entre 1000 et 31000 - dmax (place pour écrire les graduations et les projections) - 1000 (distance arbitraire entre la dernière graduation et l'intersection avec l'autre axe).
2) Il y a des valeurs négatives et positives = les limites sont 1000 et 31000. Mais il faut calculer la valeur de pos pour voir si l'intersection des deux axes n'est pas trop proche du bord du dessin, auquel cas il faudrait réduire l'intervalle utilisable de dmax pour l'écriture du texte.
Cette vérification est faite en (5) dans la procédure et nécessite l'emploi de l'aiguillage sw.
3) Les valeurs commencent par la valeur 0.
4) les valeurs sont toutes strictement positives.

****

CONVCHAR

BUT:

Conversion de valeurs en chaînes de caractères.

ACCES:

CONVCHAR(aval:rtab2;aptnbr:integer;var tchar:achar)

PARAMETRES:
---

aval : tableau des valeurs à convertir
aptnbr : nombre de valeurs dans aval
précond. : 0 < = aptnbr < = 50
tchar : tableau des chaînes de caractères résultat.

```
                postcond. : tchar[i] a une longueur de 5.

*****************************************************************

                                CONVRC
                                ------

BUT:
----
        Conversion d'un réel en une suite de caractères.

ACCES:
------
        CONVRC(nbre:real;var char:string10)

PARAMETRES:
-----------
        nbre : le réel à convertir
                précond. : -9999 < = nbre < = 9999.
        char : le résultat de la conversion
                postcond. : les 5 derniers car. de la chaîne sont
                            à blanc.

DISCUSSION:
-----------
        - Cette procédure ne sert qu'à convertir les graduations
          des axes.
        - Le point décimal et le signe comptent chacun pour un
          caractère, le signe + peut être omis ( il est alors
          remplacé par un blanc ).

PROGRAMMATION:
--------------
        * variables:
          - j est l'indice de la position à remplir dans le
            tableau de caractères.
          - Min est la valeur limite valant 10**-5 au-delà de
            laquelle on considère qu'un nombre vaut 0.
        * corps:
          Le travail se fait en trois parties:
                1 : traitement du signe
                2 : traitement de la partie entière
                3 : traitement de la partie décimale.

*****************************************************************

                                DAXEH
                                -----

BUT:
----
        Dessin d'un axe horizontal.

ACCES:
------
    DAXEH(ga:gval;andct:ndct2;posh,posv:integer;achart:achar;
          name:string10;color,style,ptnb:integer;var mf:metafile)

PARAMETRES:
```

---

ga : enregistrement
Pour la signification des variables, cf la structure de données (1.3.1.3.).
précond. : 0 < = grsize,not1,not2,not3,not4,anam, dmax < = 32767
0 < = nbpt < = 50

andct : tableau des valeurs normalisées des graduations de l'axe horizontal
précond. : 0 < = andct[i] < = 32767

posh : valeur normalisée de b dans l'équation de l'axe horizontal : y=b
précond. : 0 < = posh < = 32767

posv : valeur normalisée de b dans l'équation de l'axe vertical : x=b
précond. : 0 < = posv < = 32767

achart : tableau des graduations de l'axe sous forme de chaînes de caractères à écrire le long de l'axe
précond. : achart[i] maximum 5 caractères

name : nom de l'axe
précond. : name maximum 10 caractères

color : couleur de l'axe
précond. : 1 < = color < = 7

style : style de l'axe
précond. : 1 < = style < = 4

ptnb : nombre de graduations dans achart
précond. : 0 < = ptnb < = 50

mf : le métafichier dans lequel on écrit.

DISCUSSION:

On parle ici d'axe horizontal et vertical plutôt que d'axe x et y car il peut y avoir eu une rotation du dessin (cf mémoire, point 3.2.1)

PROGRAMMATION:

---

1) Tracé du trait représentant l'axe.
2) Détermination du côté de l'axe où sera le texte : en dessous ou au-dessus selon que l'axe est dans la moitié inférieure ou supérieure du dessin.
3) Ecriture du nom de l'axe à droite ou à gauche selon que l'axe vertical est tracé dans la moitié gauche ou droite du dessin.
4) Tracé des traits représentant les graduations.
5) Tracé des graduations elles-mêmes en une ou deux lignes selon leur nombre.

****

DAXEV

BUT:

Dessin d'un axe vertical.

ACCES:

DAXEV(ga:gval;andct:ndct2;posv,posh:integer;achart:achar;

```
            name:string10;color,style,ptnb:integer;var mf:
            metafile)
```

PARAMETRES:

```
        ga : enregistrement
                Pour la signification des variables, cf la
                structure de données ( 1.3.1.3. ).
                précond. : 0 <= grsize,not1,not2,not3,not4,anam,
                                dmax <= 32767
                           0 <= nbpt <= 50
        andct : tableau des valeurs normalisées des graduations
                de l'axe vertical
                précond. : 0 <= andct[i] <= 32767
        posv : valeur normalisée de b dans l'équation de l'axe
               vertical : x=b
                précond. : 0 <= posv <= 32767
        posh : valeur normalisée de b dans l'équation de l'axe
               horizontal : y=b
                précond. : 0 <= posh <= 32767
        achart : tableau des graduations de l'axe sous forme de
                 chaînes de caractères à écrire le long de l'axe
                précond. : achart[i] maximum 5 caractères
        name : nom de l'axe
                précond. : name maximum 10 caractères
        color : couleur de l'axe
                précond. : 1 <= color <= 7
        style : style de l'axe
                précond. : 1 <= style <= 4
        ptnb : nombre de graduations dans achart
                précond. : 0 <= ptnb <= 50
        mf : le métafichier dans lequel on écrit.
```

DISCUSSION:

On parle ici d'axe horizontal et vertical plutôt que d'axe x et y car il peut y avoir eu une rotation du dessin (cf mémoire, point 3.2.1)
Comme l'axe est vertical et que l'écriture se fait horizontalement, on n'a pas ici de système équivalent à l'affichage sur deux étages de DAXEH.

PROGRAMMATION:

1) Tracé du trait représentant l'axe.
2) Détermination du côté de l'axe où sera le texte : à droite ou à gauche selon que l'axe est dans la moitié droite ou gauche du dessin.
3) Ecriture du nom de l'axe en haut ou en bas selon que l'axe horizontal est tracé dans la moitié inférieure ou supérieure du dessin.
4) Tracé des traits représentant les graduations.
5) Tracé des graduations.

****

DETPOS

BUT:

Calcul de la coord. normalisée de l'origine de l'axe
(la graduation 0 en valeur initiale).

ACCES:

DETPOS(andct:ndct2;rwtab:rtab2;aptnb:integer;var pos:
integer;var err:boolean)

PARAMETRES:
---

andct : tableau des valeurs normalisées des graduations de l'axe
précond. : 0 < = andct[i] < = 32767
rwtab : tableau des valeurs initiales des graduations de l'axe
aptnb : nombre de valeurs dans rwtab
précond. : 0 < = aptnb < = 50
pos : valeur normalisée de l'origine
postcond. : 0 < = pos < = 32767
err : booléen
postcond. : vaut true si l'origine n'a pas pu être déterminée.

DISCUSSION:

Pour déterminer la position de l'autre axe du système, la valeur 0 doit être une valeur initiale de l'axe lorsque l'ensemble de ses valeurs passe de valeurs négatives à des valeurs positives.

CONDITION D'ERREUR:
---

0 n'est pas une valeur initiale de l'axe alors que l'ensemble de ses valeurs passe de valeurs négatives à des valeurs positives.
Message: "la grad. 0 manque sur l'axe"

****

EXPO (fonction)

BUT:

Calcul de 10 exposant i.

ACCES:

EXPO(k:integer):real

PARAMETRES:
---

k : l'indice de la puissance.

****

FCONV

BUT:

Conversion de valeurs initiales en valeurs normalisées

ACCES:

```
FCONV(frwt:rtabl;fptnb,ndcl,ndc2:integer;rwl,rw2:real;var
      fndct:ndctl)
```

PARAMETRES:

```
frwt : tableau des valeurs initiales à convertir
fptnb : le nombre de valeurs dans frwt
        précond. : 0 < = fptnb < = 500
ndcl,ndc2 : valeurs normalisées du premier (et
            respectivement dernier) point de
            l'intervalle
        précond. : 0 < = ndcl,ndc2 < = 32767
rwl,rw2 : valeurs initiales du premier (et resp.
          dernier) point de l'intervalle
        précond. : rw2 > rwl
fndct : tableau résultat contenant les valeurs
        normalisées
        postcond. : 0 < = fndct[i] < = 32767
```

PROGRAMMATION:
---

1) On choisit un intervalle de référence dont on connaît les valeurs initiales et normalisées pour le premier et le dernier point.
2) On calcule le nombre d'unités initiales et normalisées qu'il y a dans l'intervalle, puis on calcule le nombre d'unités normalisées correspondant à une unité initiale (variable tmp2) .
3) Ensuite on multiplie la distance (initiale) point à convertir moins premier point de l'intervalle par le ratio obtenu en 2) .

****

FCONV2

BUT:

Conversion de valeurs initiales en valeurs normalisées

ACCES:

```
FCONV2(frwt:rtab2;fptnb,ndcl,ndc2:integer;rwl,rw2:real;
       var fndct:ndct2)
```

PARAMETRES:

```
frwt : tableau des valeurs initiales à convertir
fptnb : le nombre de valeurs dans frwt
        précond. : 0 < = fptnb < = 50
ndcl,ndc2 : valeurs normalisées du premier (et
            respectivement dernier) point de l'intervalle
```

précond. : 0 < = ndc1,ndc2 < = 32767
rw1,rw2 : valeurs initiales du premier (et resp. dernier) point de l'intervalle
précond. : rw2 > rw1
fndct : tableau résultat contenant les valeurs normalisées
postcond. : 0 < = fndct[i] < = 32767

PROGRAMMATION:

1) On choisit un intervalle de référence dont on connaît les valeurs initiales et normalisées pour le premier et le dernier point.
2) On calcule le nombre d'unités initiales et normalisées qu'il y a dans l'intervalle, puis on calcule le nombre d'unités normalisées correspondant à une unité initiale (variable tmp2) .
3) Ensuite on multiplie la distance (initiale) point à convertir moins premier point de l'intervalle par le ratio obtenu en 2) .

****

HDRAW

BUT:

Dessin des rectangles constituant un histogramme.

ACCES:

HDRAW(ptndc,ndch:ndct2;ptnb:integer;o:opt;var mf: metafile)

PARAMETRES:

ptndc : tableau des valeurs normalisées des graduations de l'axe (limites de classe)
précond. : 0 < = ptndc[i] < = 32767
ndch : tableau des valeurs normalisées des hauteurs des rectangles
précond. : 0 < = ndch[i] < = 32767
ptnb : nombre de limites de classe
précond. : 0 < = ptnb < = 41
o : enregistrement
pour la signification des variables et leurs valeurs admissibles, cf la structure de données (1.3.1.3.) et la procédure HISTO.
précond. : toutes les valeurs sont correctes.
mf : le métafichier dans lequel on écrit

****

HVISF

BUT:

Dessin de la projection des points de la courbe choisis

sur l'axe horizontal.

ACCES:

```
HVISF(ga:gval;rwf:rtab2;ndca:ndct2;ndcfh,ndcfv:ndct2;
      fpnb,hpos:integer;var mf:metafile)
```

PARAMETRES:

ga : enregistrement
Pour la signification des variables, cf la structure de données (1.3.1.3.).
précond. : 0 < = grsize,not1,not2,not3,not4,anam, dmax < = 32767
0 < = nbpt < = 50

rwf : tableau des valeurs initiales des coordonnées verticales des points à projeter

ndca : tableau des valeurs normalisées des graduations de l'axe vertical
précond. : 0 < = ndca[i] < = 32767

ndcfh,ndcfv : tableaux des valeurs normalisées des coordonnées horizontales (respectivement verticales) des points à projeter
précond. : 0 < = ndcfh[i],ndcfv[i] < = 32767

fpnb : nombre de points à projeter
précond. : 0 < = fpnb < = 50

hpos : valeur normalisée de b dans l'équation de l'axe horizontal : z=b

mf : le métafichier dans lequel on écrit.

DISCUSSION:

On parle ici d'axe horizontal et vertical plutôt que d'axe x et y car il peut y avoir eu une rotation du dessin (cf mémoire, point 3.2.1).

PROGRAMMATION:

1) Détermination du côté de l'axe où sera le texte : en haut ou en bas selon que l'axe est dans la moitié supérieure ou inférieure du dessin.
2) Tracé des pointillés entre les point à projeter et l'axe.
   a) Test si la valeur du point à projeter n'est pas déjà égale à la valeur d'une graduation de l'axe, auquel cas on n'écrit pas une deuxième fois la même valeur le long de l'axe.
   b) Qu'il y ait inversion du dessin ou pas, les valeurs normalisées des graduations de l'axe horizontal sont toujours en ordre croissant.

****

INV

BUT:

Test d'ordre sur deux valeurs.

ACCES:

INV(nbptx,nbpty:integer;var res:boolean)

PARAMETRES:
----

nbptx : la première valeur
nbpty : la deuxième valeur
err : booléen
postcond. : vaut true si nbpty nbptx

****

MEFDH

BUT:

Construction d'une zone, l'axe x étant horizontal.

ACCES:

MEFDH(tabh,tabv:ndctl;inf,nb,posh:integer;var ptabh,
ptabv:ndctl)

PARAMETRES:
----

tabh,tabv : tableaux des valeurs normalisées des
abscisses (respectivement ordonnées) des
points de la courbe.
inf : indice (dans l'ensemble des points de la courbe)
du premier point de la courbe compris dans la zone
courante
précond. : 0 <= inf <= 500
nb : nombre de points de la courbe repris dans la zone
courante
précond. : 0 <= nb <= 500
posh : valeur normalisée de b dans l'équation de l'axe
x : y=b
précond. : 0 <= posh <= 32767
ptabh,ptabv : tableaux des valeurs normalisées des
abscisses (resp. ordonnées) des points de
la zone construite.
postcond. : 0 <= ptabh[i],ptabv[i] <= 32767.

DISCUSSION:

Une zone est formée par la courbe, l'axe x et les deux droites d'équations x=xl et x=x2, xl et x2 étant les abscisses des premier et dernier point de la courbe que l'on veut inclure dans la zone. Les points définissant la zone (ceux renvoyés dans ptabh et ptabv) sont donc les points de la courbe + le point à l'intersection de l'axe x et de la droite d'équation x = xl + le point à l'intersection de l'axe x et de la droite d'équation x=x2.

****

MEFDV

BUT:

Construction d'une zone, l'axe x étant vertical.

ACCES:

```
MEFDV(tabh,tabv:ndct1;inf,nb,posv:integer;var ptabh,
      ptabv:ndct1)
```

PARAMETRES:
---

tabh,tabv : tableaux des valeurs normalisées des abscisses (respectivement ordonnées) des points de la courbe.

inf : indice (dans l'ensemble des points de la courbe) du premier point de la courbe compris dans la zone courante
précond. : 0 < = inf < = 500

nb : nombre de points de la courbe repris dans la zone courante
précond. : 0 < = nb < = 500

posv : valeur normalisée de b dans l'équation de l'axe x x=b
précond. : 0 < = posv < = 32767

ptabh,ptabv : tableaux des valeurs normalisées des abscisses (resp. ordonnées) des points de la zone construite.
postcond. : 0 < = ptabh[i],ptabv[i] < = 32767.

DISCUSSION:

Une zone est formée par la courbe, l'axe x et les deux droites d'équations x=x1 et x=x2, x1 et x2 étant les abscisses des premier et dernier point de la courbe que l'on veut inclure dans la zone. Les points définissant la zone (ceux renvoyés dans ptabh et ptabv) sont donc les points de la courbe + le point à l'intersection de l'axe x et de la droite d'équation x = x1 + le point à l'intersection de l'axe x et de la droite d'équation x=x2.

****

MINMAX

BUT:

Recherche de la valeur et de la position du minimum et du maximum d'un ensemble de valeurs.

ACCES:

```
MINMAX(tab:rtab2;dim:integer;var min,max:real;var posmin,
       posmax:integer)
```

PARAMETRES:

```
        tab : tableau des valeurs à manipuler
        dim : nombre de valeurs dans tab
                précond. : 0 < = dim < = 50
        min,max : la valeur minimum (respectivement maximum) de
                  tab
        posmin,posmax : la position de min (resp. max) dans
                        l'ensemble des valeurs de tab
                postcond. : 0 < = posmin,posmax < = 50.

**********************************************************************

                                NUAGE
                                -----

BUT:
----
        Dessin d'un ensemble de marqueurs.

ACCES:
------
        NUAGE(var mf:metafile;a,b:ndct2;dim,couleur,mk:integer;
              nom:achar)

PARAMETRES:
-----------
        mf : le métafichier dans lequel on écrit,
        a,b : tableaux des valeurs normalisées des abscisses
              (respectivement ordonnées) des positions où il faut
              écrire les marqueurs
                précond. : 0 < = a[i],b[i] < = 32767,
        dim : le nombre de marqueurs à tracer
                précond. : 0 < = dim < = 50,
        couleur : la couleur des marqueurs
                précond. : 1 < = couleur < = 7,
        mk : le type des marqueurs
                précond. : 1< = mk < = 5,
        nom : le tableau reprenant les chaînes de caractères à
              écrire à côté de chaque marqueur.

**********************************************************************

                                PHNDC
                                -----

BUT:
----
        Calcul des valeurs normalisées des hauteurs des
        rectangles d'un histogramme.

ACCES:
------
        PHNDC(hrw:rtab2;nbpt:integer;max:real;var h2:ndct2)

PARAMETRES:
-----------
        hrw : tableau des valeurs initiales des hauteurs
        nbpt : nombre de limites de classes
                précond. : 0 < = nbpt < = 50
        max : valeur maximum de hrw
        h2 : tableau résultat des valeurs normalisées des
```

hauteurs
postcond. : 0 < = h2[i] < = 32767.

PROGRAMMATION:

Le rectangle le plus haut aura toujours une hauteur normalisée de h.

****

PHRW

BUT:

Calcul de la valeur initiale de la hauteur des rectangles d'un histogramme.

ACCES:

PHRW(pt:rtab2;nbpt:integer;r:rtab2;var h:rtab2)

PARAMETRES:
---

pt : tableau des valeurs initiales des limites de classes.
nbpt : nombre de valeurs dans pt.
précond. : 0 < = nbpt < = 50
r : tableau des répétitions
h : tableau résultat des valeurs initiales des hauteurs des rectangles.

DISCUSSION:

La hauteur égale l'aire (la valeur de la répétition) divisée par la largeur de la classe.

****

PDRAW

BUT:

Dessin de l'ensemble des zones composant la fonction.

ACCES:

PDRAW(o:opt;f:fndc;posx:integer;var mf:metafile)

PARAMETRES:
---

o : enregistrement
pour la signification des variables et leurs valeurs admissibles, cf la structure de données (4.3.1.3) et la procédure FCT.
précond. : toutes les valeurs sont correctes.
f : enregistrement comprenant
fndcx : tableau des valeurs normalisées des abscisses des points de la courbe

```
            précond. : 0 < = fndcx[i] < = 32767
        fndcy : tableau des valeurs normalisées des ordonnées
                des points de la courbe
            précond. : 0 < = fndcy[i] < = 32767
    posx : valeur normalisée de b dans l'équation de l'axe x
           : y=b
            précond. : 0 < = posx < = 32767
    mf : le métafichier dans lequel on écrit.

DISCUSSION:
-----------
    Une zone est formée par la courbe, l'axe x et les deux
    droites d'équations x=x1 et x=x2, x1 et x2 étant les
    abscisses des premier et dernier point de la courbe que
    l'on veut inclure dans la zone.

PROGRAMMATION:
--------------
    * variables :
      inf est l'indice du premier point de la zone dans
      l'ensemble des points de la courbe
    * corps :
      Le travail se fait en trois étapes :
          1 : Découpage de la courbe en autant d'ensembles
              de points qu'il y a de zones.
          2 : Rajout à tous ces ensembles des deux points
              nécessaires à l'obtention du polygone
              constituant la zone (cf les procédures MEFDH
              et MEFDV).
          3 : Dessin des zones.
      Deux lignes après (* 3 *) : on fait -1 parce que le
      dernier point de la zone appartient aussi à la zone
      suivante.

**********************************************************************

                                POLYGO
                                ------

BUT:
----
    Dessin d'un polygone.

ACCES:
------
    POLYGO(var mf:metafile;a,b:ndctl;dim,couleur,lst,ist,fc,
           fi:integer;nom:string10)

PARAMETRES:
-----------
    mf : le métafichier dans lequel on écrit,
    a,b : tableaux des valeurs normalisées des abscisses
          (respectivement ordonnées) des points du bord du
          polygone
            précond. : 0 < = a[i],b[i] < = 32767,
    dim : le nombre de points définissant le bord du polygone
            précond. : 0 < = dim < = 500,
    couleur : la couleur du bord du polygone
            précond. : 1 < = couleur < = 7,
    lst : le style du bord du polygone
```

```
            précond. : 1 < = lst < = 4,
        ist : le type de l'intérieur du polygone
            précond. : 0  < = ist < = 1,
        fc : la couleur de l'intérieur du polygone
            précond. : 1 < = fc < = 7,
        fi : le style de hachurage de l'intérieur du polygone
            précond. : 1 < = fi < = 7,
        nom : le nom du polygone.

*****************************************************************
```

PROJC

BUT:

Recherche des valeurs initiales et normalisées des points à projeter d'une courbe.

ACCES:

```
PROJC(projtab:ndct2;frwx,frwy:rtabl;fndcx,fndcy:ndctl;nb:
      integer var rrwtx,rrwty:rtab2;var rndctx,rndcty:
      ndct2)
```

PARAMETRES:

projtab : tableau des indices (dans l'ensemble des points de la courbe) des points à projeter
précond. : 1 < = projtab[i] < = nombre de points de la courbe

frwx,frwy : tableaux des valeurs initiales des abscisses (respectivement ordonnées) des points de la courbe

fndcx,fndcy : tableaux des valeurs normalisées des abscisses (resp. ordonnées) des points de la courbe
précond. : 0 < = fndcx[i],fndcy[i] < = 32767

nb : nombre de valeurs dans projtab
précond. : 0 < = nb < = 50

rrwtx,rrwty : tableaux des valeurs initiales des abscisses (resp. ordonnées) des points de la courbe à projeter

rndctx,rndcty : tableaux des valeurs normalisées des abscisses (resp. ordonnées) des points de la courbe à projeter
précond. : 0 < = rndctx[i],rndcty[i] < = 32767

DISCUSSION:

On extrait des ensembles des valeurs initiales et normalisées des points de la courbe les valeurs initiales et normalisées des points que l'on veut projeter, à partir des indices des points à projeter dans l'ensemble des points de la courbe.

```
*****************************************************************
```

TEXT

BUT:

Fixation des attributs du texte d'un dessin.

ACCES:

TEXT(size:integer;var mf:metafile)

PARAMETRES:
---

size : taille des caractères
mf : le métafichier dans lequel on écrit.

DISCUSSION:

Si la variable size = 1, on a choisi la petite taille de caractères sinon c'est la grande taille.
On fixe ici pour le métafichier courant la taille des caractères, l'espacement inter-caractères (espacement standard, c'est à dire celui utilisé pour ce texte-ci), le type des caractères (lettres rectangulaires) et la précision du texte (string precision).

****

VERACP

BUT:

Vérification de la validité des options choisies par l'utilisateur.

ACCES:

VERACP(t1,t2:rtab2;vnb,inb:integer;o:opt;ccol,vcol,icol:
integer;var error:boolean)

PARAMETRES:
---

t1,t2 : tableaux des corrélations variables-axes principaux
postcond. : les erreurs sont détectées
vnb,inb : nombre des variables (respectivement individus)
postcond. : 0 < = vnb,inb < = 50
o : enregistrement
Pour la signification des variables, les contraintes existantes sur leurs valeurs et le traitement des erreurs éventuelles, cf la structure de données (1.3.1.3.) et la procédure ACP.
postcond. : les variables ont des valeurs correctes.
ccol,vcol,icol : couleur du cercle, des variables, des individus sur le dessin
postcond. : 0 < = ccol,vcol,icol < = 7
error : booléen
vaut true si une erreur grave a été détectée.

CONDITION D'ERREUR:

1) vnb ou inb non compris entre 1 et 50,
   message : " le nbre des var. ou des individus n'est pas compris entre 1 et 50 "
2) une corrélation (un élément de t1 ou t2) non comprise entre -1 et 1,
   message : " une corrélation a une valeur non comprise entre -1 et 1 "

****

VERAX

BUT:

Vérification de la croissance stricte des graduations de l'axe x.

ACCES:

VERAX(typ,axnb:integer;ax:rtab2;var error:boolean)

PARAMETRES:

typ : type de l'axe x
axnb : nombre de valeurs dans ax (cf ci-dessous)
   précond. : 0 < = axnb < = 50
ax : tableau des valeurs initiales des graduations de l'axe x
error : booléen
   postcond. : vaut true si on n'a pas la croissance stricte.

DISCUSSION:

Si la variable typ = 1 l'axe x est numérique, sinon il est alphanumérique.
Pour les contraintes existantes sur l'axe x, cf la procédure FCT.

CONDITION D'ERREUR:

Les graduations de l'axe x ne sont pas strictement croissantes.
Message : " grad de l'axe x non strct >. ".

PROGRAMMATION:

La vérification doit se faire uniquement si l'axe est numérique, d'où la présence du paramètre typ.

****

VERAY

BUT:

Vérification de la croissance stricte des graduations de l'axe y.

ACCES:

VERAY(aynb:integer;ay:rtab2;var error:boolean)

PARAMETRES:
---

aynb : nombre de valeurs dans ay (cf ci-dessous)
précond. : 0 < = aynb < = 50
ay : tableau des valeurs initiales des graduations de l'axe y
error : booléen
postcond. : vaut true si on n'a pas la croissance stricte.

DISCUSSION:

Pour les contraintes existantes sur l'axe y, cf la procédure FCT.

CONDITION D'ERREUR:

Les graduations de l'axe y ne sont pas strictement croissantes.
Message : " grad de l'axe y non strct >. ".

****

VEROPT

BUT:

Vérification de la validité des options choisies par l'utilisateur.

ACCES:

VEROPT(o:opt;var error:boolean)

PARAMETRES:
---

o : enregistrement.
Pour la signification des variables,les contraintes existantes sur leurs valeurs et le traitement des erreurs éventuelles, cf la structure de données (1.3.1.3.) et la procédure FCT
postcond. : le traitement des erreurs est réalisé
error : booléen
postcond. : vaut true si une erreur grave a été commise.

DISCUSSION:

En plus des vérifications individuelles sur chaque variable, on vérifie également si les zones que l'on a définies reprennent bien tous les points de la courbe. Si ce n'est pas le cas, la découpe en zones est ignorée

(on ne dessine qu'une seule zone). Ce fait est signalé par le message " la découpe de la fct en zones étant incorrecte, elle est ignorée. ".

CONDITION D'ERREUR:

1) axnb,aynb non compris entre 2 et 50,
2) fnb non compris entre 2 et 500,
3) une valeur de z[i].dim non comprise entre 0 et 500.
   Message : " valeur non permise pour une option. "

PROGRAMMATION:

La variable total représente le nombre de points de la courbe déjà repris dans les zones sauf le dernier à droite car ce dernier sera repris dans la zone suivante. D'où également l'instruction total:= total+1 par la suite.

****

VERPOS

BUT:

Vérification que des valeurs sont positives.

ACCES:

VERPOS(tab:rtab2;nb:integer;var err:boolean)

PARAMETRES:

tab : tableau des valeurs à tester
nb : nombre de valeurs dans tab
   précond. : 0 <= nb <= 50
err : booléen
   postcond. : vaut true si une erreur est détectée.

CONDITION D'ERREUR:

Une des valeurs au moins est strictement négative.

****

VERPRO

BUT:

Vérification de la validité des indices des points de la courbe que l'on veut projeter sur les axes.

ACCES:

VERPRO(pnbre,fnbre:integer;p:ndct2;var error:boolean)

PARAMETRES:

pnbre : nombre de points de la courbe à projeter
précond. : 0 < = pnbre < = 50
fnbre : nombre de points de la courbe
précond. : 0 < = fnbre < = 500
p : tableau des indices des points de la courbe que l'on veut projeter
error : booléen
postcond. : vaut true si une erreur est détectée

DISCUSSION:

On considère que l'ensemble des points de la courbe est numéroté de 1 à fnbre.

CONDITION D'ERREUR:

Un des indices est négatif ou plus grand que fnbre.
Message : " numéro de proj inexistant. ".

****

VVISF

BUT:

Dessin de la projection des points de la courbe choisis sur l'axe vertical.

ACCES:

```
VVISF(ga:gval;rwf:rtab2;ndca:ndct2;ndcfh,ndcfv:ndct2;
      fpnb,vpos:integer;var mf:metafile)
```

PARAMETRES:
---

ga : enregistrement
Pour la signification des variables, cf la structure de données (1.3.1.3.).
précond. : 0 < = grsize,not1,not2,not3,not4,anam, dmax < = 32767
0 < = nbpt < = 50
rwf : tableau des valeurs initiales des coordonnées verticales des points à projeter
ndca : tableau des valeurs normalisées des graduations de l'axe vertical
précond. : 0 < = ndca[i] < = 32767
ndcfh,ndcfv : tableaux des valeurs normalisées des coordonnées horizontales (respectivement verticales) des points à projeter
précond. : 0 < = ndcfh[i],ndcfv[i] < = 32767
fpnb : nombre de points à projeter
précond. : 0 < = fpnb < = 50
vpos : valeur normalisée de b dans l'équation de l'axe vertical : z=b
mf : le métafichier dans lequel on écrit.

DISCUSSION:

On parle ici d'axe horizontal et vertical plutôt que

d'axe x et y car il peut y avoir eu une rotation du dessin (cf mémoire, point 3.2.1)

PROGRAMMATION:

1) Détermination du côté de l'axe où sera le texte : droite ou gauche selon que l'axe est dans la moitié droite ou gauche du dessin.
2) Tracé des pointillés entre les points à projeter et l'axe.
   a) Test si la valeur du point à projeter n'est pas déjà égale à la valeur d'une graduation de l'axe, auquel cas on n'écrit pas une deuxième fois la même valeur le long de l'axe.
   b) Il n'y a pas d'inversion du dessin; l'axe vertical est l'axe y du dessin.
   c) Il y a eu inversion du dessin; l'axe vertical est l'axe x du dessin et par rapport au point b) les valeurs normalisées des graduations sont en ordre décroissant (au lieu de croissant), d'où le test f>a au lieu de f<a en b).

****

WCLIP

BUT:

Vérification de la valeur de données.

ACCES:

```
WCLIP(min,max:real;fptnb:integer;var f:rtabl;var err:
     boolean)
```

PARAMETRES:

```
min,max : les valeurs-limites inférieure (et
          respectivement supérieure)
fptnb : le nombre des données dans f
        précond. : 0 <= fptnb <=500
f : le tableau des données à vérifier
        postcond. : min <= f[i] <= max
err : booléen
        postcond. : vaut true si au moins une des données
                    a dû être rectifiée.
```

DISCUSSION:

Lorsque la valeur d'une des données de f est inférieure (supérieure) au minimum (maximum), cette donnée prend la valeur du minimum (maximum).

****

CHAPTER 2
PARTIE FORTRAN.

2.1. INTRODUCTION.

Cette partie est divisée en trois:

- les considérations générales relatives à l'ensemble de la partie Fortran,
- les procédures de représentation et de manipulation de graphes,
- les procédures relatives aux histogrammes et aux classifications hiérarchiques.

2.2. CONSIDERATIONS GENERALES RELATIVES A L'ENSEMBLE DES PROCEDURES.

2.2.1. PROGRAMMATION : DECOUPE EN NIVEAUX

L'ensemble de la librairie Fortran est constitué de deux types de procédures.

D'une part, les procédures d'application qui sont destinées a être appelées dans le programme d'application écrit par l'utilisateur.

D'autre part, les sous-procédures des précédentes.

2.2.2. CONTROLE DES ERREURS.

La validité des paramètres est vérifiée dans les procédures d'application. Dans ce cas, elles correspondent à des fonctions entières (integer function) et le code d'erreur (éventuel) est renvoyé dans le nom de la fonction. L'utilisateur doit tenir compte de ce fait pour réaliser l'appel correct à de telles procédures.

Les procédures pour lesquelles aucun contrôle n'est effectué correspondent à des sous-programmes (subroutine).

2.2.3. NOTATIONS.

Le terme " code graphique " est utilisé pour désigner un nombre entier positif de 4 chiffres où chaque chiffre désigne une propriété graphique d'un objet, à savoir:

- premier chiffre = la couleur:

* sur le GIGI	rouge = 1 vert = 2 jaune = 3 bleu = 4	magenta = 5 cyan = 6 blanc = 7 noir = 8
* sur la table	rouge = 1 vert = 2	noir = 3 bleu = 4

- second chiffre = le style :

* sur le GIGI	tirets = 1 axes = 2	pointillé = 3 solide = 4
* sur la table	solide = 1 pointillé = 2	tirets = 3 solide = 4

- troisième chiffre = la largeur de trait:
  fin = 1
  moyen = 2
  large = 3

- quatrième chiffre = la brillance du trait:
  faible = 1
  moyenne = 2
  forte = 3

A titre d'exemple, le code 2433 désigne un tracé solide de couleur verte et de largeur et brillance maximales.

Par défaut, le code graphique vaut 1422.

Le nom d'un métafichier est une chaîne de 10 caractères au maximum .

Par défaut, ce nom est "DI3000.MTA".

2.2.4. NOMS RESERVES.

L'ensemble des procédures utilisent des zones communes (COMMON en Fortran) pour la communication interne de certaines informations. Ces zones possèdent des noms réservés de la forme:

AREAn où n varie entre 1 et 11 .

L'utilisateur ne peut donc pas utiliser ces noms pour étiqueter ses propres zones internes.

De même, les procédures de manipulation de graphes utilisent les segments retenus portant les numeros 1 et 32000. L'utilisateur ne peut donc créer de segments retenus portant ces numeros.

2.3. LES PROCEDURES DE REPRESENTATION ET DE MANIPULATION DE GRAPHES.

2.3.1. PRELIMINAIRES.

2.3.1.1. La zone graphique.

Comme nous l'avons indiqué dans le mémoire (chapitre 3, section 4), la représentation des graphes est basée sur une structure interne appelée la zone graphique.

Cette zone est constituée de deux tableaux, GSAREA et GAAREA .

GSAREA est un tableau [50,9] d'entiers ou GSAREA(i,j) contient la j ème propriété graphique du sommet de numero interne i .

	signification	plage de val.	val. par defaut
GSAREA(i,1)	couleur	1 à 8	1 à 8 ($\neq$ de la couleur de fond)
2	style	1 à 4	4 (solide)
3	largeur trait	0 à 32767	16383
4	brillance	0 à 32767	16383
5	num. du sommet	0 à 99	X
6	pointeur détail	qcq	X
7	statut	-1 à 1	X
8	indice-ligne	1 à 9	X
9	indice-colonne	1 à 11	X

GAAREA est un tableau [250,9] d'entiers ou GAAREA(i,j) contient la j ème propriété graphique de l'arc de numero interne i .

	signification	plage de val.	val. par defaut
GAAREA(i,1)	couleur	1 à 8	1 à 8 ($\neq$ de la couleur de fond)
2	style	1 à 4	4 (solide)
3	largeur trait	0 à 32767	16383
4	brillance	0 à 32767	16383
5	num. de l'arc	0 à 999	X

6	pointeur détail	qcq	X
7	statut	-1 à 1	X
8	num. sommet init.	1 à 9	X
9	num. sommet term.	1 à 11	X

Une croix (X) dans le tableau signifie qu'il n'existe pas de valeur par defaut .

Dans les spécifications des procédures, un paramètre est considéré comme invalide s'il n'appartient pas à la plage de valeurs définies dans les tableaux ci-dessus.

Le statut indique l'état dans lequel se trouve le sommet :

-1 si le sommet n'a pas encore été créé
(initialement)

0 si le sommet appartient à un segment temporaire
(avant validation)

1 si le sommet appartient au segment retenu
(après validation)

Le statut est sous la responsabilité du système. Les autres éléments sont fournis par l'utilisateur à l'aide des procédures décrites ci-après (point 2.3.2).

2.3.1.2. Séquence des appels.

Pour assurer une exécution correcte de son programme, l'utilisateur doit respecter une séquence dans les appels aux procédures d'application. Cette séquence est décrite par l'organigramme ci-dessous.

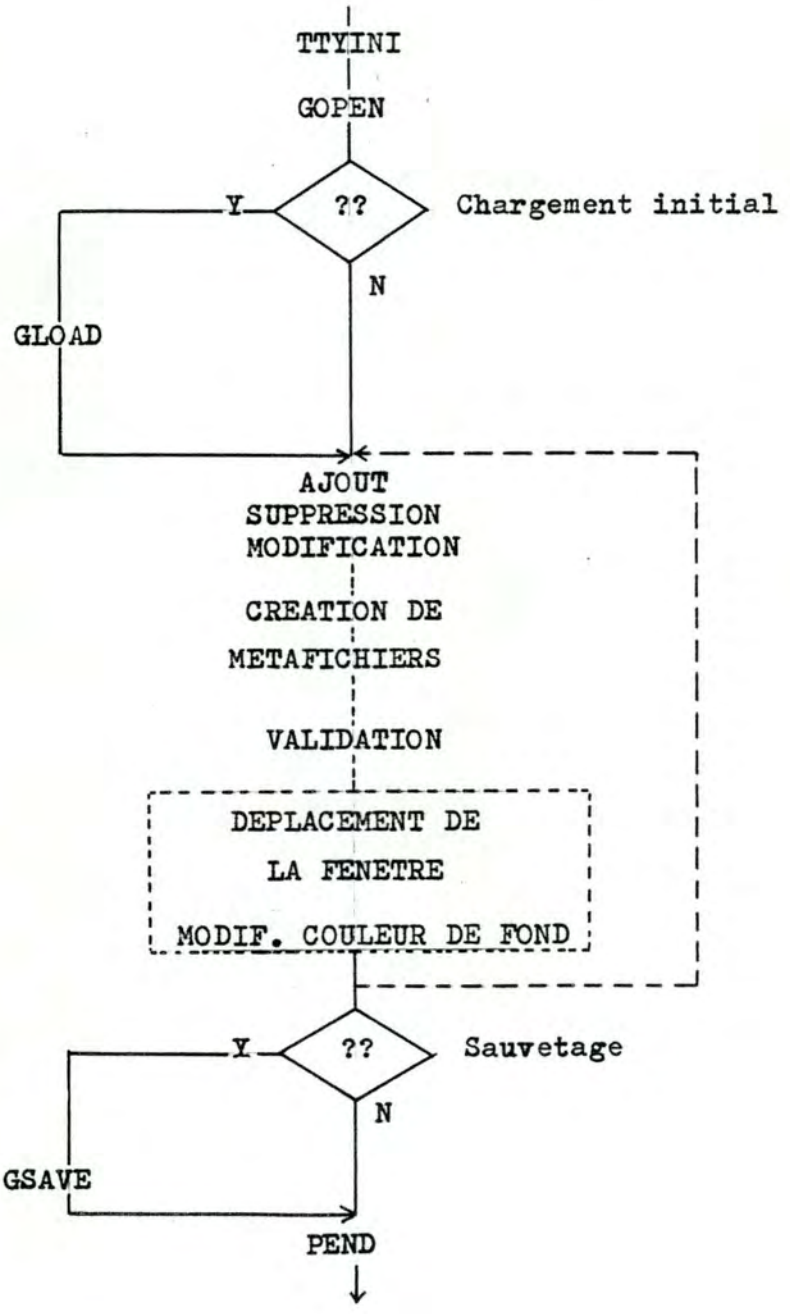
L'appel à TTYINI permet d'initialiser un terminal autre que le terminal graphique afin de permettre une manipulation interactive du graphe.

L'appel à GOPEN doit apparaître avant tout appel aux procédures (autres que TTYINI). De même, l'appel à PEND doit clôturer toute manipulation graphique.

Une modification de la couleur de fond ou un déplacement de la fenêtre doit être précédé d'un appel à PVALID.

2.3.1.3. Exemple d'utilisation.

On trouve ici la liste des appels nécessaires pour produire le graphe présenté comme premier exemple de réalisation dans le mémoire (chapitre 3, section 4).



```
      program graphe


     -- init. du programme d'application--

      logical orient
      double precision mfname
      integer gdev

     -- lecture des données --
              .
              .
              .
              .
      mfname='graphe.mta'        ! nom du métafichier
      orient=.true.              ! graphe orienté
      gdev=1                     ! device graphique GIGI
      ns=8                       ! 8 sommets
      na=10                      ! 10 arcs

      call GOPEN(gdev,orient)

      do 1000 i=1,ns,1

      itmpl=ADDSOM(nums(i),pnts(i),cods(i),il(i),ic(i))

      if(itmpl.ne.0) goto 3000    ! erreur lors de
                                  ! l'ajout d'un sommet
1000  continue

      do 2000 i=1,na,1

      itmpl=ADDARC(numa(i),pnta(i),coda(i),si(i),st(i))

      if(itmpl.ne.0) goto 3000     ! erreur lors de
                                   ! l'ajout d'un arc
2000  continue

      call PPRINT(mfname)          ! création d'un
                                   ! métafichier
3000  call PEND
      stop
      end
```

On obtient le même résultat en utilisant GLOAD qui charge l'ensemble du graphe en une seule fois.

On peut également construire le graphe interactivement en positionnant un à un les sommets et les arcs (après initialisation éventuelle d'un autre terminal via TTYINI).

2.3.2. LES PROCEDURES.

Les procédures sont classées dans l'ordre suivant :

1. Procédures d'application, c'est à dire celles qui peuvent être appelées directement par l'utilisateur.
2. Procédures de mise à jour de la zone graphique et de l'image.
3. Procédures utilitaires.

Dans chaque groupe, les procédures sont classées par ordre alphabétique.

2.3.2.1. Procédures d'application.

****

ADDARC

BUT:

Ajoute un arc au graphe.

ACCES:

ADDARC(NUMARC,PNTR,CODE,SI,STT)

PARAMETRES:

NUMARC [integer;input]

- numéro de l'arc à ajouter

PNTR [integer;input]

- pointeur détail associé à l'arc

CODE [integer;input]

- code graphique de l'arc

SI [integer;input]

- numéro du sommet initial

STT [integer;input]

- numéro du sommet terminal

DISCUSSION:

Après vérification des contraintes sur les paramètres, ADDARC permet d'ajouter un arc sur l'écran et dans la

zone graphique.

CONDITION D'ERREUR:

ADDARC = -1 si la zone graphique relative aux arcs est remplie,
ou si le numéro numarc est invalide,
ou s'il existe déjà un arc portant le numéro numarc,

= -2 si le sommet initial ou terminal n'existe pas,
ou si la position désignée par ces deux sommets est déjà occupée.

****

ADDSOM

BUT:

Ajoute un sommet au graphe.

ACCES:

ADDSOM(NUMSOM,PNTR,CODE,ILINE,ICOL)

PARAMETRES:

NUMSOM [integer;input]

- numéro du sommet à ajouter

PNTR [integer;input]

- pointeur détail associé au sommet

CODE [integer;input]

- code graphique du sommet

ILINE [integer;input]

- indice ligne de la position du sommet

ICOL [integer;input]

- indice colonne de la position du sommet

DISCUSSION:

Après vérification des contraintes sur les paramètres, ADDSOM permet d'ajouter un sommet sur l½écran et dans la zone graphique.

CONDITION D'ERREUR:

ADDSOM = -1 si la zone graphique relative aux sommets est remplie,

ou si le numéro numsom est invalide,
ou s'il existe déjà un sommet portant le numéro numsom,

= -2 si les indices ligne et colonne sont invalides,
ou si la position [iline,icol] est déjà occupée.

****

DELARC

BUT:

Détruit un arc du graphe.

ACCES:

DELARC(NUMARC)

PARAMETRES:

NUMARC [integer;input]

- numéro de l'arc à détruire

DISCUSSION:

DELARC permet de détruire un arc sur l'écran et dans la zone graphique.

CONDITION D'ERREUR:

DELARC = -1 s'il n'existe pas d'arc portant le numéro numarc.

****

DELSOM

BUT:

Détruit un sommet du graphe.

ACCES:

DELSOM(NUMSOM)

PARAMETRES:

NUMSOM [integer;input]

- numéro du sommet à détruire

DISCUSSION:

DELSOM permet de détruire un sommet et les arcs incidents

à ce sommet, sur l'écran et dans la zone graphique.

CONDITION D'ERREUR:

DELSOM = -1 s'il n'existe pas de sommet portant le numéro numsom.

****

GLOAD

BUT:

Permet l'affichage d'un graphe à partir d'une zone de données contenant les propriétes graphiques des sommets et des arcs.

ACCES:

GLOAD(ENSS,ENSA,NS,NA)

PARAMETRES:

ENSS [integer;array;input]

- tableau [1..ns;1..3] contenant les propriétés graphiques des sommets

ENSA [integer;array;input]

- tableau [1..na;1..5] contenant les propriétés graphiques des arcs

NS [integer;input]

- nombre maximum de positions de sommets que l'on veut afficher

NA [integer;input]

- nombre d'arcs à afficher

DISCUSSION:

Le tableau enss a le format suivant :

enss(i,1) = numéro du i ème sommet

enss(i,2) = pointeur détail associé

enss(i,3) = code graphique

Le tableau ensa a le format suivant :

ensa(i,1) = numéro du i ème arc

ensa(i,2) = pointeur détail associé

ensa(i,3) = code graphique

ensa(i,4) = numéro du sommet initial

ensa(i,5) = numéro du sommet terminal

Les sommets sont positionnés séquentiellement dans la grille de référence DE HAUT EN BAS et DE GAUCHE A DROITE aux intersections des lignes et colonnes d'indice pair et des lignes et colonnes d'indice impair. Ceci n'assure donc pas nécessairement une répartition " élégante des sommets dans le plan de vision.

S'il le désire, l'utilisateur a la possibilité de fournir le tableau enss de telle façon que certaines positions de la grille soient evitées (skipped) . En effet, si le numéro du i ème sommet (enss(i,1)) est négatif, GLOAD laissera la i ème position de la grille inoccupée (La i ème position est calculée en fonction de l'ordre dans lequel on attribue ces positions. C'est ce qui justifie que ns définisse le nombre max. de positions à afficher et non pas le nombre de sommets).

Ainsi,l'utilisateur a la possibilité d'afficher un graphe en une seule fois moyennant le formattage des données d'entrée. Ce graphe initial peut ensuite être modifié avant d'être sauvé (à l'aide de GSAVE) pour un rechargement ultérieur.

Quant aux arcs, ils sont bien sûr placés automatiquement en fonction des numéros de sommet initial et terminal.

CONDITION D'ERREUR:

GLOAD = -1 si la capacité de la zone graphique relative aux sommets a été dépassée lors du chargement, ou si au moins une donnée relative aux sommets est erronée .

= -2 si la capacité de la zone graphique relative aux arcs a été dépassée lors du chargement, ou si au moins une donnée relative aux arcs est erronée .

****

GOPEN

BUT:

Initialise DI-3000 et les paramètres nécessaires à l'affichage du graphe.

ACCES:

CALL GOPEN(DEVICE,ORI)

PARAMETRES:

DEVICE [integer;input]

- numéro de l'appareil graphique à initialiser.

ORI [logical;input]

- variable booléenne qui vaut true si le graphe est orienté, et qui vaut false sinon

DISCUSSION:

Le paramètre device désigne le numéro logique de l'appareil à initialiser. Ce numéro dépend de l'environnement graphique dans lequel on travaille.

Dans le cas du terminal GIGI, ce numéro vaut 1.

Cette procédure doit être appelée avant toute autre procédure de manipulation du graphe.
(Voir aussi 2.3.1.2.)

****

GRAPNR

BUT:

Renvoie le pointeur détail d'un arc.

ACCES:

GRAPNR(NUMARC,PNTR)

PARAMETRES:

NUMARC [integer;input]

- numéro de l'arc dont on veut lire le pointeur détail

PNTR [integer;output]

- valeur du pointeur détail

CONDITION D'ERREUR:

GRAPNR = -1 s'il n'existe pas d'arc portant le numéro numarc.

****

GRSPNR

BUT:

Renvoie le pointeur détail d'un sommet.

ACCES:

GRSPNR(NUMSOM,PNTR)

PARAMETRES:

NUMSOM [integer;input]

- numéro du sommet dont on veut lire le pointeur détail

PNTR [integer;output]

- valeur du pointeur détail

CONDITION D'ERREUR:

GRSPNR = -1 s'il n'existe pas de sommet portant le numéro numsom.

****

GSAVE

BUT:

Sauve le graphe couramment construit dans une zone de données contenant les propriétés graphiques des sommets et des arcs.

ACCES:

GSAVE(ENSS,ENSA,NS,NA)

PARAMETRES:

ENSS [integer;array;output]

- tableau [1..ns;1..3] contenant les propriétés graphiques des sommets

ENSA [integer;array;output]

- tableau [1..na;1..5] contenant les propriétés graphiques des arcs

NS [integer;input]

- nombre maximum de positions de sommets que l'on veut sauver

NA [integer;input]

- nombre d'arcs à afficher

DISCUSSION:

Le tableau enss a le format suivant :

enss(i,1) = numéro du i ème sommet

enss(i,2) = pointeur détail associé

enss(i,3) = code graphique

Le tableau ensa a le format suivant :

ensa(i,1) = numéro du i ème arc

ensa(i,2) = pointeur détail associé

ensa(i,3) = code graphique

ensa(i,4) = numéro du sommet initial

ensa(i,5) = numéro du sommet terminal

DISCUSSION:

Cette procédure réalise l'effet inverse de GLOAD. Les positions des sommets sont comptées de la même façon et les tableaux qui constituent les sorties de GSAVE sont ceux qui constituent les entrées de GLOAD.

CONDITION D'ERREUR:

GSAVE = -1 si ns est supérieur au nombre maximum de sommets que l'on peut afficher,
ou si na est supérieur au nombre d'arcs que l'on peut afficher.

****

GWAPNR

BUT:

Modifie le pointeur détail d'un arc.

ACCES:

GWAPNR(NUMARC,PNTR)

PARAMETRES:

NUMARC [integer;input]

- numéro de l'arc dont on veut modifier le pointeur détail

PNTR [integer;input]

- valeur du nouveau pointeur détail

DISCUSSION:

GWAPNR permet de modifier le chaînage entre la zone graphique relative aux arcs et une autre zone de données éventuelle.

CONDITION D'ERREUR:

GWAPNR = -1 s'il n'existe pas d'arc portant le numéro numarc.

****

GWSPNR

BUT:

Modifie le pointeur détail d'un sommet.

ACCES:

GWSPNR(NUMSOM,PNTR)

PARAMETRES:

NUMSOM [integer;input]

- numéro du sommet dont on veut modifier le pointeur détail

PNTR [integer;input]

- valeur du pointeur détail

DISCUSSION:

GWSPNR permet de modifier le chaînage entre la zone graphique relative aux sommets et une autre zone de zone de données éventuelle.

CONDITION D'ERREUR:

GWSPNR = -1 s'il n'existe pas de sommet portant le numéro numsom.

****

MDCARC

BUT:

Modifie le code graphique d'un arc.

ACCES:

MDCARC(NUMARC,CODE)

PARAMETRES:

NUMARC [integer;input]

- numéro de l'arc à modifier

CODE [integer;input]

- nouveau code graphique

DISCUSSION:

MDCARC modifie le code de l'arc sur l'écran et dans la zone graphique. L'arc est d'abord effacé puis retracé avec ses attributs graphiques.

CONDITION D'ERREUR:

MDCARC = -1 s'il n'existe pas d'arc portant le numéro numarc.

****

MDCSOM

BUT:

Modifie le code graphique d'un sommet.

ACCES:

MDCSOM(NUMSOM,CODE)

PARAMETRES:

NUMSOM [integer;input]

- numéro du sommet à modifier

CODE [integer;input]

- nouveau code graphique

DISCUSSION:

MDCSOM modifie le code du sommet sur l'écran et dans la zone graphique.Le sommet est d'abord effacé puis retracé avec ses attributs graphiques.

CONDITION D'ERREUR:

MDCSOM = -1 s'il n'existe pas de sommet portant le numero numsom.

****

MDNARC

BUT:

Modifie le numéro d'un arc.

ACCES:

MDNARC(NUMARC,NEWN)

PARAMETRES:

NUMARC [integer;input]

- numéro de l'arc à modifier

NEWN [integer;input]

- nouveau numero de l'arc

DISCUSSION:

MDNARC modifie le numéro de l'arc sur l'écran et dans la zone graphique. L'arc est d'abord effacé puis retracé avec ses attributs graphiques.

CONDITION D'ERREUR:

MDNARC = -1 s'il n'existe pas d'arc portant le numéro numarc,

-2 si newn n'est pas compris entre 0 et 999, ou s'il existe déjà un arc portant ce numero.

****

MDNSOM

BUT:

Modifie le numéro d'un sommet.

ACCES:

MDNSOM(NUMSOM,NEWN)

PARAMETRES:

NUMSOM [integer;input]

- numéro du sommet à modifier

NEWN [integer;input]

- nouveau numéro du sommet

DISCUSSION:

MDNSOM modifie le numéro du sommet sur l'écran et dans la zone graphique. Le sommet est d'abord effacé puis retracé avec ses attributs graphiques.

CONDITION D'ERREUR:

MDNSOM = -1 s'il n'existe pas de sommet portant le numéro numsom,

-2 si newn n'est pas compris entre 0 et 99, ou s'il existe déjà un sommet portant ce

numéro.

****

MDPARC

BUT:

Modifie la position d'un arc.

ACCES:

MDPARC(NUMARC,NEWSI,NEWST)

PARAMETRES:

NUMARC [integer;input]

- numéro de l'arc à modifier

NEWSI [integer;input]

- nouveau numéro de sommet initial

NEWST [integer;input]

- nouveau numéro de sommet terminal

DISCUSSION:

MDPARC modifie la position de l'arc sur l'écran et dans la zone graphique.L'arc est d'abord effacé puis retracé avec ses attributs graphiques.

CONDITION D'ERREUR:

MDPARC = -1 s'il n'existe pas d'arc portant le numéro numarc,

-2 si le nouveau sommet initial ou terminal n'existe pas,
ou si la position designée par ces deux sommets est déjà occupée.

****

MDPSOM

BUT:

Modifie la position d'un sommet.

ACCES:

MDPSOM(NUMSOM,ILINE,ICOL)

PARAMETRES:

NUMSOM [integer;input]

- numéro du sommet à modifier

ILINE [integer;input]

- nouvel indice ligne de la position

ICOL [integer;input]

- nouvel indice colonne de la position

DISCUSSION:

MDPSOM modifie la position du sommet sur l'écran et dans la zone graphique. Le sommet est d'abord effacé puis retracé avec ses attributs graphiques.

CONDITION D'ERREUR:

MDPSOM = -1 s'il n'existe pas de sommet portant le numéro numsom,

-2 si iline ou icol est invalide,
ou si iline et icol ne sont pas tous les deux pairs ou tous les deux impairs,
ou si la position [iline,icol] est déjà occupée.

****

MOVEGR

BUT:

Positionne la fenêtre dans le plan de vision.

ACCES:

MOVEGR(NUMGR)

PARAMETRES:

NUMGR [integer;input]

- numéro de la fenêtre que l'on veut positionner
précond: 1 <= numgr <= 6

DISCUSSION:

Le graphe est decoupé en 6 zones, chaque zone correspondant à une fenêtre. Le schéma de la découpe a été donné dans le mémoire au chapitre 3, section 4.

La sixième zone correspond à l'ensemble du graphe.

Les valeurs de numgr sont les suivantes:

1 = fenêtre inférieure gauche

2 = fenêtre supérieure gauche
3 = fenêtre supérieure droite
4 = fenêtre inférieure droite
5 = centre du graphe
6 = ensemble du graphe

CONDITION D'ERREUR:

MOVEGR = -1 si le numéro numgr n'est pas compris entre 1 et 6.

PROGRAMMATION:

Les paramètres de la fenêtre sont d'abord positionnés, puis la mise à l'échelle et la translation s'effectuent.

****

PBACKG

BUT:

Modifie la couleur de fond.

ACCES:

PBACKG(COLOR)

PARAMETRES:

COLOR [integer;input]

- numéro de la nouvelle couleur de fond

DISCUSSION:

Les numéros des couleurs disponibles sont donnés au point 2.2.3

L'appel à cette procédure doit être précédé d'un appel à PVALID (Voir aussi le point 2.3.1.2).

CONDITION D'ERREUR:

PBACKG = -1 si le numéro de couleur est erroné.

****

PEND

BUT:

Assure la terminaison correcte des procédures d'affichage.

ACCES:

CALL PEND

DISCUSSION:

Cette procédure termine la séquence des appels. Elle doit être la dernière à être appelée dans le programme d'application de l'utilisateur.

****

PPRINT

BUT:

Crée un métafichier contenant le graphe couramment construit.

ACCES:

CALL PPRINT(MFNAME)

PARAMETRES:

MFNAME [double precision;input]

- nom du métafichier que l'on crée

DISCUSSION:

Seul le graphe est dessiné dans le métafichier. La grille de référence n'y apparaît plus.

PROGRAMMATION:

Le métafichier est créé à partir des informations memorisées dans la zone graphique.

****

PVALID

BUT:

Valide le graphe couramment créé.

ACCES:

CALL PVALID

DISCUSSION:

Le problème de la validation a été discuté dans le mémoire au chapitre 3, section 4.

Pour rappel, cette procédure doit être appelée avant toute modification de la couleur de fond ou tout déplacement de la fenêtre.

PROGRAMMATION:

La variable status vaut 0 si le sommet ou l'arc est dans un segment temporaire. Après validation elle vaut 1 pour indiquer que le sommet ou l'arc se trouve dans le segment retenu.

****

TTYINI

BUT:

Initialise le terminal d'entrées-sorties.

ACCES:

CALL TTYINI(DEVICE)

PARAMETRES:

DEVICE [integer;char.string;input]

- chaîne de 5 caractères au maximum contenant le nom logique du terminal

DISCUSSION:

TTYINI permet d'initialiser le dialogue entre le programme appelant et un terminal de nom logique device (ex: "TTY20") autre que le terminal graphique afin d'éviter sur ce dernier la surimpression des dessins et des messages d'entrées-sorties.

PROGRAMMATION:

STTY est une procédure Assembleur qui permet le déplacement arrière du curseur sur le terminal d'entrées-sorties.

****

2.3.2.2. Procédures de mise à jour de la zone graphique et de l'image.

****

GADARC

BUT:

Ajoute un arc, en zone graphique.

ACCES:

CALL GADARC(NUMARC,PNTR,CODE,SI,STT)

PARAMETRES:

NUMARC [integer;input]

- numéro de l'arc à ajouter

PNTR [integer;input]

- pointeur détail associé à l'arc

CODE [integer;input]

- code graphique de l'arc

SI [integer;input]

- numéro du sommet initial

STT [integer;input]

- numéro du sommet terminal

PROGRAMMATION:

On attribue à l'arc la première entrée libre de la zone graphique relative aux arcs.
La variable nac compte le nombre d'arcs couramment créés.

****

GADSOM

BUT:

Ajoute un sommet, en zone graphique.

ACCES:

CALL GADSOM(NUMSOM,PNTR,CODE,ILINE,ICOL)

PARAMETRES:

NUMSOM [integer;input]

- numéro du sommet à ajouter

PNTR [integer;input]

- pointeur détail associé au sommet

CODE [integer;input]

- code graphique du sommet

ILINE [integer;input]

- indice ligne de la position du sommet

ICOL [integer;input]

- indice colonne de la position du sommet

PROGRAMMATION:

On attribue au sommet la première entrée libre de la zone graphique relative aux sommets.
La variable nsc compte le nombre de sommets couramment créés.

****

GDLARC

BUT:

Détruit un arc, en zone graphique.

ACCES:

CALL GDLARC(NINARC)

PARAMETRES:

NINARC [integer;input]

- numéro interne de l'arc
précond: 1 < = ninarc < = 250

PROGRAMMATION:

La variable nac compte le nombre d'arcs couramment créés.

****

GDLSOM

BUT:

Détruit un sommet, en zone graphique.

ACCES:

CALL GDLSOM(NINSOM)

PARAMETRES:

NINSOM [integer;input)

- numéro interne du sommet
précond: 1 < = ninsom < = 50

DISCUSSION:

GDLSOM détruit également, en zone graphique, les arcs incidents au sommet.

PROGRAMMATION:

La variable nsc compte le nombre de sommets couramment créés.

****

GINIT

BUT:

Initialise la zone graphique relative à un graphe.

ACCES:

CALL GINIT(ORI)

PARAMETRES:

ORI [logical;input]

- variable booléenne qui vaut true si le graphe est orienté, et qui vaut false sinon

PROGRAMMATION:

La variable nsc compte le nombre de sommets couramment créés.

La variable nac compte le nombre d'arcs couramment créés.

****

GMAARC

BUT:

Modifie le code graphique d'un arc, en zone graphique.

ACCES:

CALL GMAARC(NINARC,CODE)

PARAMETRES:

NINARC [integer;input]

- numéro interne de l'arc
  précond: 1 < = ninarc < = 250

CODE [integer;input]

- nouveau code graphique

****

GMASOM

BUT:

Modifie le code graphique d'un sommet, en zone graphique.

ACCES:

CALL GMASOM(NINSOM,CODE)

PARAMETRES:

NINSOM [integer;input]

- numéro interne du sommet
précond: 1 < = ninsom < = 50

CODE [integer;input]

- nouveau code graphique

****

GMNARC

BUT:

Modifie le numéro d'un arc, en zone graphique.

ACCES:

CALL GMNARC(NINARC,NEWN)

PARAMETRES:

NINARC [integer;input]

- numéro interne de l'arc
précond: 1 < = ninarc < = 250

NEWN [integer;input]

- nouveau numéro de l'arc

****

GMNSOM

BUT:

Modifie le numéro d'un sommet, en zone graphique.

ACCES:

CALL GMNSOM(NINSOM,NEWN)

PARAMETRES:

NINSOM [integer;input]

- numéro interne du sommet
précond: 1 < = ninsom < = 50

NEWN [integer;input]

- nouveau numéro du sommet

****

GMPARC

BUT:

Modifie la position d'un arc, en zone graphique.

ACCES:

CALL GMPARC(NINARC,NEWSI,NEWST)

PARAMETRES:

NINARC [integer;input]

- numéro interne de l'arc
précond: 1 < = ninarc < = 250

NEWSI [integer;input]

- nouveau numéro de sommet initial

NEWST [integer;input]

- nouveau numéro de sommet terminal

****

GMPSOM

BUT:

Modifie la position d'un sommet, en zone graphique.

ACCES:

CALL GMPSOM(NINSOM,ILINE,ICOL)

PARAMETRES:

NINSOM [integer;input]

- numéro interne du sommet
précond: 1 < = ninsom < = 50

ILINE [integer;input]

- nouvel indice ligne de la position

ICOL [integer;input]

- nouvel indice colonne de la position

****

PADARC

BUT:

Affiche l'image d'un arc.

ACCES:

CALL PADARC(NINARC)

PARAMETRES:

NINARC [integer;input]

- numéro interne de l'arc
précond: 1 < = ninarc < = 250

DISCUSSION:

L'arc est représenté par un trait joignant le sommet initial au sommet terminal. Une boucle est representée par un cercle accolé au coin supérieur droit du sommet correspondant.

Le signe d'orientation (éventuel) apparaît à mi-distance entre les sommets initial et terminal.

Le numéro de l'arc est dessiné à mi-distance entre le signe d'orientation et le sommet terminal.

Tous les élements nécessaires à l'affichage sont tirés de la zone graphique.

PROGRAMMATION:

La variable scalfa est le facteur d'échelle. Elle vaut 2 pour un tracé dans une fenêtre avec agrandissement, et vaut 1 pour un tracé normal.

La variable booléenne orient vaut true si le graphe est orienté, et vaut false sinon.

Pour tracer un arc, on distingue plusieurs cas dépendant de la position des sommets initial et terminal.
Dans chaque cas, on détermine la position du signe d'orientation.

cas 0 : tracé d'une boucle

cas 1 a 4 : tracé d'un arc

cas 1 : les sommets initial et terminal sont alignés verticalement, le sommet terminal est vers le bas.

cas 2 : les sommets initial et terminal sont alignés verticalement, le sommet terminal est vers le haut.

cas 3 : les sommets initial et terminal sont disposés de façon quelconque, le sommet terminal est le plus à gauche.

cas 4 : les sommets initial et terminal sont disposés de façon quelconque, le sommet terminal est le plus à droite.

****

PADSOM

BUT:

Affiche l'image d'un sommet.

ACCES:

CALL PADSOM(NINSOM)

PARAMETRES:

NINSOM [integer;input]

- numéro interne du sommet
précond: 1 < = ninsom < = 50

DISCUSSION:

Le sommet est représenté par un carré au centre duquel apparaît le numéro du sommet.

Tous les éléments nécessaires à l'affichage sont tirés de la zone graphique.

PROGRAMMATION:

Les coordonnées du centre du carré représentant le sommet sont calculées à partir des indices ligne et colonne, en fonction de la fenêtre courante.

La variable scalfa est le facteur d'échelle. Elle vaut 2 pour un tracé dans une fenêtre avec agrandissement, et vaut 1 pour un tracé normal.

****

PDLARC

BUT:

Détruit l'image d'un arc.

ACCES:

CALL PDLARC(NINARC)

PARAMETRES:

NINARC [integer;input]

- numéro interne de l'arc
  précond: 1 < = ninarc < = 250

PROGRAMMATION:

La destruction consiste à retracer l'arc dans la couleur de fond courante.

****

PDLSOM

BUT:

Détruit l'image d'un sommet.

ACCES:

CALL PDLSOM(NINSOM)

PARAMETRES:

NINSOM [integer;input]

- numéro interne du sommet
  précond: 1 < = ninsom < = 50

DISCUSSION:

PDLSOM détruit également l'image des arcs incidents au sommet.

PROGRAMMATION:

La destruction consiste à retracer le sommet et les arcs incidents dans la couleur de fond courante.

****

PGRILL

BUT:

Dessine la grille de référence pour le positionnement des sommets du graphe.

ACCES:

CALL PGRILL(NL,NC)

PARAMETRES:

NL [integer;input]

- nombre de lignes de la grille de référence

NC [integer;input]

- nombre de colonnes de la grille de référence

DISCUSSION:

La grille est constituée d'un cadre gradué en abscisses et en ordonnées à concurrence du nombre de colonnes et et de lignes.

Cette grille est affichée lors de l'initialisation du graphe par la procédure GOPEN.

PROGRAMMATION:

La grille correspond au segment retenu numéro 1.

****

PINIT

BUT:

Initialise l'image pour l'affichage du graphe.

ACCES:

CALL PINIT(DEVICE)

PARAMETRES:

DEVICE [integer;input]

- numéro de l'appareil graphique à initialiser.

DISCUSSION:

Le paramètre device désigne le numéro logique de l'appareil à initialiser. Ce numéro dépend de l'environnement graphique dans lequel on travaille.

Dans le cas du terminal GIGI, ce numéro vaut 1.

PINIT affiche également la grille de référence dans laquelle on positionne les sommets du graphe.

PROGRAMMATION:

La routine JFILES oriente les messages d'erreur provenant

de DI-3000 vers le fichier FOR01.DAT dans le catalogue de l'utilisateur.

Le tableau Ginfo contient les éléments nécessaires à la manipulation du graphe (déplacements de la fenêtre).

Le segment 32000 contient l'image du graphe. Il est initialement créé vide.

****

PMOVGR

BUT:

Mise à l'échelle et translation du graphe.

ACCES:

CALL PMOVGR

DISCUSSION:

PMOVGR met à l'échelle et translate les deux segments retenus contenant la grille de référence et le graphe.

Cela produit un effet d'agrandissement de la partie du dessin sélectionnée à l'aide des paramètres d'affichage.

****

2.3.2.3. Procédures utilitaires.

****

GADECO

BUT:

Décode le code graphique.

ACCES:

CALL GADECO(CODE,CL,ST,LT,BR)

PARAMETRES:

CODE [integer;input]

- code graphique

CL [integer;output]

- numéro de couleur
  postcond: 1 <= cl <= 8

ST [integer;output]

- numéro de style
  postcond: 1 <= st <= 4

LT [integer;output]

- indice de largeur de trait
  postcond: 0 <= lt <= 32767

BR [integer;output]

- indice de brillance du trait
  postcond: 0 <= br <= 32767

DISCUSSION:

La structure du code graphique a été donnée au point 2.2.3 .

PROGRAMMATION:

Les valeurs du 3 ème et du 4 ème chiffre sont utilisées pour calculer des indices variant entre 0 et 32767.

****

GANUMI

BUT:

Renvoie le numéro interne d'un arc dont on fournit le numéro utilisateur.

ACCES:

GANUMI(NUMARC,NINARC)

PARAMETRES:

NUMARC [integer;input]

- numéro utilisateur de l'arc

NINARC [integer;output]

- numéro interne de l'arc
  postcond: 1 <= ninarc <= 250

CONDITION D'ERREUR:

GANUMI = -1 s'il n'existe pas d'arc portant le numéro numarc.

****

GENCOD

BUT:

Convertit un nombre entier positif de 3 chiffres maximum en une chaîne de caractères.

ACCES:

CALL GENCOD(NUM,L,STRING)

PARAMETRES:

NUM [integer;input]

- numéro à convertir
  précond: 0 < num <= 999

L [integer;output]

- longueur de la chaîne de caractères
  précond: 1 <= 1 <= 3

STRING [integer;char.string;output]

- chaîne de caractères

DISCUSSION:

La paramètre string contient des blancs si num n'est pas compris entre 0 et 999.

****

GMOVGR

BUT:

Positionne les paramètres de la fenêtre d'affichage.

ACCES:

CALL GMOVGR(SCALE,TX,TY)

PARAMETRES:

SCALE [real;input]

- facteur d'agrandissement
  précond: 0 <= scale <= 100

TX,TY [real;input]

- composantes de la translation

DISCUSSION:

Les composantes de la translation sont exprimées en coordonnées virtuelles. Pour cette raison, elles ne peuvent excéder la hauteur et la largeur de l'espace des coordonnées virtuelles.

****

GPDECO

BUT:

Vérifie la validité de la position d'un sommet.

ACCES:

GPDECO(ILINE,ICOL)

PARAMETRES:

ILINE [integer;input]

- indice-ligne de la position du sommet

ICOL [integer;input]

- indice-colonne de la position du sommet

CONDITION D'ERREUR:

GPDECO = -1 si l'indice-ligne ou colonne n'est pas valide ou si iline et icol ne sont pas tous les deux pairs ou tous les deux impairs,

-2 si la position [iline,icol] est déjà attribuée.

****

GPENCO

BUT:

Encode les valeurs des indices ligne et colonne de la grille de référence.

ACCES:

CALL GPENCO(NUML,NUMC,IL,IC)

PARAMETRES:

NUML [integer;input]

- nombre de lignes de la grille de référence

NUMC [integer;input]

- nombre de colonnes de la grille de référence

IL [integer;array;output]

- tableau [1..50] des indices ligne

IC [integer;array;output]

- tableau [1..50] des indices colonne

DISCUSSION:

GPENCO crée deux tableaux d'indices il et ic, tels que la paire [il(i),ic(i)] corresponde à la position du i ème sommet dans l'ordre d'apparition des sommets.

Sur le dessin, ces positions sont attribuées aux sommets de haut en bas et de gauche a droite.

****

GSNUMI

BUT:

Renvoie le numéro interne d'un sommet dont on fournit le numéro utilisateur.

ACCES:

GSNUMI(NUMSOM,NINSOM)

PARAMETRES:

NUMSOM [integer;input]

- numéro utilisateur du sommet

NINSOM [integer;output]

- numéro interne du sommet
  postcond: 1 < = ninsom < = 50

CONDITION D'ERREUR:

GSNUMI = -1 s'il n'existe pas de sommet portant le numéro numsom.

****

PCONVP

BUT:

Renvoie les coordonnées réelles d'un sommet en fonction de la fenêtre courante.

ACCES:

CALL PCONVP(NINSOM,XPOS,YPOS)

PARAMETRES:

NINSOM [integer;input]

- numéro interne du sommet
  précond: 1 < = ninsom < = 50

XPOS [real;output]

- abscisse de la coordonnée du sommet

YPOS [real;output]

- ordonnée de la coordonnée du sommet

DISCUSSION:

PCONVP convertit les indices ligne et colonne d'un sommet en ses coordonnées réelles.

Les coordonnées dépendent de la fenêtre courante (état agrandi ou non).

PROGRAMMATION:
---

1. Conversion des coordonnées réelles en coordonnées virtuelles.

2. Calcul des nouvelles coordonnées virtuelles en fonction de l'échelle et de la translation.

3. Conversion des coordonnées virtuelles en coordonnées reelles.

****

2.4. LES PROCEDURES RELATIVES AUX CLASSIFICATIONS HIERARCHIQUES ET AUX HISTOGRAMMES.

2.4.1. PRELIMINAIRES.

2.4.1.1. Notations.

Le terme " code graphique " est utilisé pour désigner un nombre entier positif de 4 chiffres où chaque chiffre désigne une propriété graphique d'un objet, à savoir:

- premier chiffre = la couleur:

* sur le GIGI	rouge = 1	magenta = 5
	vert = 2	cyan = 6
	jaune = 3	blanc = 7
	bleu = 4	noir = 8
* sur la table	rouge = 1	noir = 3
	vert = 2	bleu = 4

- second chiffre = le style :

* sur le GIGI	tirets = 1	pointillé = 3
	axes = 2	solide = 4
* sur la table	solide = 1	tirets = 3
	pointillé = 2	solide = 4

- troisième chiffre = la largeur de trait:

fin = 1
moyen = 2
large = 3

- quatrième chiffre = la brillance du trait:

faible = 1
moyenne = 2
forte = 3

A titre d'exemple, le code 2433 désigne un tracé solide de couleur verte et de largeur et brillance maximales.

Par défaut, le code graphique vaut 1422.

Le terme " code graphique de l'intérieur " est utilisé pour désigner un nombre entier positif de 2 chiffres où :

- premier chiffre = la couleur:

	rouge = 1	magenta = 5
	vert = 2	cyan = 6
	jaune = 3	blanc = 7
	bleu = 4	noir = 8

- second chiffre = le style de hachures

vertical = 1
horizontal = 2

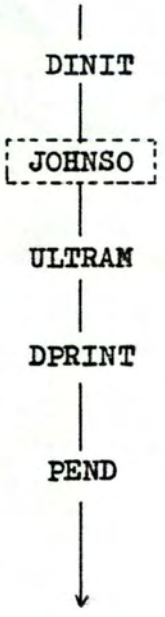
oblique droite = 3
oblique gauche = 4
style 1 et 2 = 5
style 3 et 4 = 6
plein = 7

Par défaut, le code graphique de l'intérieur vaut 11.

2.4.1.2. Séquence des appels.

Pour assurer une exécution correcte de son programme, l'utilisateur doit respecter une séquence dans les appels aux procédures d'application.

a. Classifications hiérarchiques.



b. Histogrammes.

L'appel à DINIT doit apparaître avant tout appel aux procédures. De même, l'appel à PEND doit clôturer tout affichage graphique.

2.4.2. LES PROCEDURES.

Les procédures sont classées dans l'ordre suivant :

1. Procédures d'application, c'est à dire celles qui peuvent être appelées directement par l'utilisateur.
2. Sous-procédures.

Dans chaque groupe, les procédures sont classées par ordre alphabétique.

2.4.2.1. Procédures d'application.

****

DINIT

BUT:

Initialise DI-3000 et les paramètres d'affichage.

ACCES:

CALL DINIT(DEVICE)

PARAMETRES:

DEVICE [integer;input]

- numéro de l'appareil graphique à initialiser.

DISCUSSION:

Le paramètre device désigne le numéro logique de l'appareil à initialiser. Ce numéro dépend de l'environnement graphique dans lequel on travaille.

Dans le cas du terminal GIGI, ce numéro vaut 1.

PROGRAMMATION:

La routine JFILES oriente les messages d'erreur provenant de DI-3000 vers le fichier FOR01.DAT dans le catalogue de l'utilisateur.

****

DPRINT

BUT:

Crée un métafichier contenant le dendrogramme couramment construit.

ACCES:

CALL DPRINT(MFNAME)

PARAMETRES:

MFNAME [double precision;input]

- nom du métafichier que l'on crée

DISCUSSION:

Cette procédure ne peut apparaître seule dans un programme d'application. Pour l'utiliser, il faut respecter une séquence d'appels (cfr point 2.4.1.2).

PROGRAMMATION:

Le métafichier est créé à partir des informations mémorisées dans une zone interne lors de l'appel à ULTRAM.

****

HISTO

BUT:

Dessine un histogramme.

ACCES:

HISTO(REP,LIM,N,IND,ECODE,ICODE,AXNAME,AXCODE,PCNAME)

PARAMETRES:

REP [integer;array;input]

- tableau [1..n] des répétitions par classe
  précond: rep(i) > 0

LIM [real;array;input]

- tableau [1..n + 1] des limites de classes
  précond: lim(i) strict. croissantes.

N [integer;input]

- nombre de classes

IND [integer;input]

- type de l'intérieur. Cette variable vaut 1 si l'intérieur des rectangles doit être hachuré et vaut 0 s'il doit être laissé vide

ECODE [integer;array;input]

- tableau [1..n] des codes graphiques du bord des rectangles.(Le style des bords est toujours le trait plein)

ICODE [integer;array;input]

- tableau [1..n] des codes graphiques de l'intérieur des rectangles

AXNAME [integer;array of char.string;input]

- tableau [1..2] contenant le nom de l'axe. Chaque élément de axname est une chaîne de 5 car. maximum

AXCODE [integer;input]

- code graphique de l'axe

PCNAME [integer;array of char.string;input]

- tableau [1..2] contenant le nom du dessin. Chaque élément de pcname est une chaîne de 5 car. maximum

DISCUSSION:

Les valeurs possibles pour le code intérieur des rectangles de l'histogramme sont données en 2.4.1.1 .

Les valeurs des répétitions par classe sont projetées à gauche du dessin, sur une ligne verticale.

Les graduations de l'axe unique de l'histogramme, (qui correspondent aux limites de classes) sont tracées suivant le format Fortran F5.2 . L'utilisateur doit tenir compte de ce fait dans la façon dont il interprète le dessin.

Cette procédure ne peut apparaître seule dans un programme d'application. Pour l'utiliser, il faut respecter une séquence d'appels. (cfr point 2.4.1.2)

CONDITION D'ERREUR:

HISTO = -1 si le nombre de classes n'est pas compris entre 1 et 40,

= -2 si au moins une des répétitions par classe est négative ou nulle,

= -3 si les limites de classe ne sont pas strictement croissantes.

PROGRAMMATION:

L'algorithme se déroule de la façon suivante :

ECODE [integer;array;input]

- tableau [1..n] des codes graphiques du bord des rectangles.(Le style des bords est toujours le trait plein)

ICODE [integer;array;input]

- tableau [1..n] des codes graphiques de l'intérieur des rectangles

AXNAME [integer;array of char.string;input]

- tableau [1..2] contenant le nom de l'axe. Chaque élément de axname est une chaîne de 5 car. maximum

AXCODE [integer;input]

- code graphique de l'axe

PCNAME [integer;array of char.string;input]

- tableau [1..2] contenant le nom du dessin. Chaque élément de pcname est une chaîne de 5 car. maximum

DISCUSSION:

Les valeurs possibles pour le code intérieur des rectangles de l'histogramme sont données en 2.4.1.1 .

Les valeurs des répétitions par classe sont projetées à gauche du dessin, sur une ligne verticale.

Les graduations de l'axe unique de l'histogramme, (qui correspondent aux limites de classes) sont tracées suivant le format Fortran F5.2 . L'utilisateur doit tenir compte de ce fait dans la façon dont il interprète le dessin.

Cette procédure ne peut apparaître seule dans un programme d'application. Pour l'utiliser, il faut respecter une séquence d'appels. (cfr point 2.4.1.2)

CONDITION D'ERREUR:

HISTO = -1 si le nombre de classes n'est pas compris entre 1 et 40,

= -2 si au moins une des répétitions par classe est négative ou nulle,

= -3 si les limites de classe ne sont pas strictement croissantes.

PROGRAMMATION:

L'algorithme se déroule de la façon suivante :

- vérification des contraintes
- calcul des hauteurs des classes
- tracé de l'histogramme
- conversion des limites en graduations
- tracé de l'axe et du titre
- affectation des valeurs courantes aux variables utilisées pour le tracé dans un métafichier

****

HPRINT

BUT:

Crée un métafichier contenant l'histogramme couramment construit.

ACCES:

CALL HPRINT(MFNAME)

PARAMETRES:

MFNAME [double precision;input]

- nom du métafichier que l'on crée

DISCUSSION:

Cette procédure ne peut apparaître seule dans un programme d'application. Pour l'utiliser, il faut respecter une séquence d'appels (cfr point 2.4.1.2).

PROGRAMMATION:

Le métafichier est créé à partir des informations mémorisées dans une zone interne lors de l'appel à HISTO.

****

JOHNSO

BUT:

Crée une ultramétrique par la méthode de Johnson.

ACCES:

CALL JOHNSO(D,ND,NF,IND)

PARAMETRES:

D [real;array;input et output]

- en input: tableau [1..nd,1..nd] des distances (similarités) entre individus. La distance entre l'individu i et l'individu j est donnée par d(i,j)
  précond: d est supratriangulaire

- en output: tableau [1..nd,1..nd] de l'ultramétrique
  postcond: d est supratriangulaire

ND [integer;input]

- nombre d'individus
  précond: 2 <= nd <= 40

NF [integer;input]

- numéro de la formule à utiliser pour le calcul de la distance (similarité) entre groupes.
  précond: 1 <= nf <= 3

IND [logical;input]

- variable booléenne qui vaut:
  true si les d(i,j) sont des distances
  false si les d(i,j) sont des similarités

DISCUSSION:

A chaque itération de la méthode de Johnson, on recheche les deux groupes les plus proches et on les fusionne pour former un nouveau groupe.

Pour calculer la distance entre ce nouveau groupe et les autres groupes, on peut choisir entre trois formules :

max. de la distance entre chaque groupe restant et les sous-groupes du nouveau groupe formé (nf = 1)

min. de cette distance (nf = 2)

moyenne de cette distance (nf = 3)

Par défaut, la valeur de nf est 3.

****

ULTRAM

BUT:

Dessine le dendrogramme relatif à une classification hiérarchique.

ACCES:

ULTRAM(TAB,N,IND,PCNAME,AXCODE,PCCODE)

PARAMETRES:

TAB [real;array,input]

- tableau [1..n,1..n] de l'ultramétrique
  précond: d est supratriangulaire

N [integer;input]

- nombre d'individus
  precond: 2 <= n <= 40

IND [logical;input]

- variable booléenne qui vaut:
  true si les d(i,j) sont des distances
  false si les d(i,j) sont des similarités

PCNAME [integer;array of char.string;input]

- tableau [1..2] contenant le nom du dessin. Chaque élément de pcname est une chaîne de 5 car. maximum.

AXCODE [integer;input]

- code graphique des axes

PCCODE [integer;input]

- code graphique du dendrogramme

DISCUSSION:

Les axes ne portent pas de nom et sont tracés tous deux en utilisant le même code graphique.

Les graduations le long de l'axe des ordonnées (qui correspondent aux indices d'agrégation des différents groupes) sont tracées suivant le format Fortran F5.2 . L'utilisateur doit tenir compte de ce fait dans la façon dont il interprète le dessin.

Cette procédure ne peut apparaître seule dans un programme d'application. Pour l'utiliser, il faut respecter une séquence d'appels (cfr point 2.4.1.2).

CONDITION D'ERREUR:

ULTRAM = -1 si le nombre d'individus n'est pas compris entre 2 et 40,

= -2 si le tableau tab n'est pas supratriangulaire

PROGRAMMATION:

L'algorithme se déroule de la façon suivante :

- vérification des contraintes

- classification des individus
- calcul des coordonnées des groupes
- conversion des graduations
- tracé des axes
- tracé du dendrogramme
- tracé du titre
- affectation des valeurs courantes aux variables utilisées pour le tracé dans un métafichier

****

PEND

BUT:

Assure la terminaison correcte des procédures d'affichage.

ACCES:

CALL PEND

DISCUSSION:

Cette procédure termine la séquence des appels. Elle doit être la dernière à être appelée dans le programme d'application de l'utilisateur.(Voir aussi 2.4.1.2)

****

TTYINI

BUT:

Initialise le terminal d'entrées-sorties.

ACCES:

CALL TTYINI(DEVICE)

PARAMETRES:

DEVICE [integer;char.string;input]

- chaîne de 5 caractères au maximum contenant le nom logique du terminal

DISCUSSION:

TTYINI permet d'initialiser le dialogue entre le programme appelant et un terminal de nom logique device (ex: "TTY20") autre que le terminal graphique

afin d'éviter sur ce dernier la surimpression des dessins et des messages d'entrées-sorties.

PROGRAMMATION:

STTY est une procédure Assembleur qui permet le déplacement arrière du curseur sur le terminal d'entrées-sorties.

****

2.4.2.2. Sous-procédures.

****

DAXEH

BUT:

Trace un axe horizontal.

ACCES:

```
CALL DAXEH(ABSI,NABS,GRD,AXNAME,AXCODE,DIR,NC,
           XSIZE,YSIZE)
```

PARAMETRES:

ABSI [real;array;input]

- tableau [1..nabs] des abscisses des graduations
  précond: 0 < absi(i) < xsize

NABS [integer;input]

- nombre d'abscisses

GRD [integer;array of char.string;input]

- tableau [1..nabs] des graduations. Chaque élément de grd est une chaîne de 5 car. maximum

AXNAME [integer;array of char.string;input]

- tableau [1..2] contenant le nom de l'axe.Chaque élément de axname est une chaîne de 5 car. maximum

AXCODE [integer;input]

- code graphique de l'axe

DIR [integer;input]

- direction de tracé des graduations
  précond: 1 <= dir <= 4

NC [integer;input]

- nombre de caractères à afficher pour les graduations
precond: 1 < = nc < = 5

XSIZE [real;input]

- nombre d'unités de la fenêtre de vision en abscisses

YSIZE [real;input]

- nombre d'unités de la fenêtre de vision en ordonnées

DISCUSSION:

L'axe est toujours tracé dans une fenêtre s'étendant de 0 à xsize en abscisses et de 0 à ysize en ordonnées.

Le nom de l'axe apparaît à son extrémité droite.

Les graduations sont tracées dans une des 4 directions suivantes:

de gauche à droite	(dir = 1)
de haut en bas	(dir = 2)
de droite à gauche	(dir = 3)
de bas en haut	(dir = 4)

****

DAXEV

BUT:

Trace un axe vertical.

ACCES:

```
CALL DAXEV(ORD,NORD,GRD,AXNAME,AXCODE,DIR,NC,
           XSIZE,YSIZE)
```

PARAMETRES:

ORD [real;array;input]

- tableau [1..nord] des ordonnées des graduations
précond: 0 < ord(i) < ysize

NORD [integer;input]

- nombre d'ordonnées

GRD [integer;array of char.string;input]

- tableau [1..nabs] des graduations. Chaque élément de grd est une chaîne de 5 car. maximum

AXNAME [integer;array of char.string;input]

- tableau [1..2] contenant le nom de l'axe.Chaque élément de axname est une chaîne de 5 car. maximum

AXCODE [integer;input]

- code graphique de l'axe

DIR [integer;input]

- direction de tracé des graduations
  précond: 1 < = dir < = 4

NC [integer;input]

- nombre de caractères à afficher pour les graduations
  précond: 1 < = nc < = 5

XSIZE [real;input]

- nombre d'unités de la fenêtre de vision en abscisses

YSIZE [real;input]

- nombre d'unités de la fenêtre de vision en ordonnées

DISCUSSION:

L'axe est toujours tracé dans une fenêtre s'étendant de 0 à xsize en abscisses et de 0 à ysize en ordonnées.

Le nom de l'axe apparaît à son extrémité supérieure.

Les graduations sont tracées dans une des 4 directions suivantes:

de gauche à droite	(dir = 1)
de haut en bas	(dir = 2)
de droite à gauche	(dir = 3)
de bas en haut	(dir = 4)

****

DCALC

BUT:

Calcule les coordonnées réelles des groupes composant un dendrogramme à partir de l'arbre binaire correspondant.

ACCES:

```
CALL DCALC(TREE,N,IDA,INDIV,NCI,XPOS,YPOS,XPOSG,
           XPOSD,YPOSG,YPOSD,XSIZE,YSIZE)
```

PARAMETRES:

TREE [integer;array;input]

- tableau [1..n,1..3] contenant la représentation de l'arbre binaire
  précond: l'arbre est correctement construit

N [integer;input]

- nombre de cellules de l'arbre
  précond: 1 <= n <= 79

IDA [real;array;input]

- tableau [1..nci-1] des indices d'agrégation des groupes formés

INDIV [integer;array;input]

- tableau [1..nci] des numéros des individus
  précond: indiv(i) sont les valeurs des feuilles de l'arbre et sont classées dans cet ordre

NCI [integer;input]

- nombre d'individus
  précond: 2 <= nci <= 40

XPOS [real;array;output]

- tableau [1..nci] des abscisses des individus
  postcond: 0 < xpos(i) < xsize

YPOS [real;array;output]

- tableau [1..nci-1] des ordonnées des groupes
  postcond: 0 < ypos(i) < ysize
  ypos(i) sont classés dans l'ordre croissant

XPOSG,XPOSD [real;array;output]

- tableaux [1..nci-1] des abscisses gauche et droite, respectivement, des groupes constituant le dendrogramme

YPOSG,YPOSD [real;array;output]

- tableaux [1..nci-1] des ordonnées gauche et droite, respectivement, des groupes constituant le dendrogramme

XSIZE [real;input]

- nombre d'unités de la fenêtre de vision en abscisses

YSIZE [real;input]

- nombre d'unités de la fenêtre de vision en ordonnées

DISCUSSION:

Chaque groupe du dendrogramme est représenté par deux lignes verticales et une ligne horizontale formant ainsi un U renversé.

Les variables xposg(i) et xposd(i) sont les abscisses des barres verticales. Les variables yposg(i) et yposd(i) sont les ordonnées des sous-groupes gauche et droit du

groupe i.

****

DCLASS

BUT:

Construit l'arbre binaire correspondant à une ultramétrique et renvoie les indices d'agrégation des groupements effectués.

ACCES:

```
CALL DCLASS(D,ND,IND,TREE,NB,YNOM)
```

PARAMETRES:

D [real;array,input]

- tableau [1..nd,1..nd] de l'ultramétrique
  précond: d est supratriangulaire

ND [integer;input]

- nombre d'individus
  précond: 2 <= nd <= 40

IND [logical;input]

- variable booléenne qui vaut:
  true si les d(i,j) sont des distances
  false si les d(i,j) sont des similarités

TREE [integer;array;output]

- tableau [1..nb,1..3] contenant la représentation de l'arbre binaire

NB [integer;input]

- nombre de cellules de l'arbre
  précond: nb = 2*nd -1

YNOM [real;output]

- tableau [1..nd-1] des indices d'agrégation des groupes formés

DISCUSSION:

A partir de l'ultramétrique, on crée un arbre binaire représenté par un tableau à nb lignes et 3 colonnes.

Chaque ligne correspond à une cellule de l'arbre.
Initialement, chaque cellule correspond à un individu.
A chaque groupement,on ajoute une cellule et on lui donne pour valeur le numéro du groupe formé.
On relève en même temps l'indice d'agrégation de ce

groupe.

L'adresse de la racine de l'arbre est nb (adresse de la dernière cellule).

PROGRAMMATION:

Chaque cellule i comprend 3 champs:

le champ 'gauché [tree(i,1)] contient l'adresse du sous-arbre de gauche de la cellule i

le champ 'valeur' [tree(i,2)] contient la valeur de la cellule

le champ 'droité [tree(i,3)] contient l'adresse du sous-arbre de droite de la cellule i

Chaque cellule i est soit un noeud de l'arbre, soit une feuille. Dans ce dernier cas, le champ 'gauché est nul et le champ 'droité contient l'adresse du successeur dans l'ordre infixé.

****

DCONV

BUT:

Convertit les coordonnées réelles des groupes d'un dendrogramme en chaînes de caractères pour les graduations.

ACCES:

```
CALL DCONV(IDA,INDIV,N,XPOS,YPOS,ABSI,ORD,NORD,GRDX,GRDY)
```

PARAMETRES:

IDA [real;array;input]

- tableau [1..n-1] des indices d'agrégation des groupes formés

INDIV [integer;array;input]

- tableau [1..n] des numéros des individus
  précond: indiv(i) sont les valeurs des feuilles de l'arbre de classification et sont classées dans cet ordre

N [integer;input]

- nombre d'individus
  precond: 2 < = n < = 40

XPOS [real;array;input]

- tableau [1..n] des abscisses des individus

YPOS [real;array;input]

- tableau [1..n-1] des ordonnées des groupes
précond: ypos(i) sont classés dans
l'ordre croissant

ABSI [real;array;output]

- tableau [1..n] des abscisses des graduations
de l'axe horizontal

ORD [real;array;output]

- tableau [1..nord] des ordonnées des graduations
de l'axe vertical

NORD [integer;output]

- nombre de graduations de l'axe vertical

GRDX [integer;array of char.string;output]

- tableau [1..n] des graduations de l'axe horizontal.

GRDY [integer;array of char.string;output]

- tableau [1..nord] des graduations de l'axe vertical.

DISCUSSION:

Le nombre de graduations sur l'axe vertical est différent du nombre d'indices d'agrégation étant donné que plusieurs groupes peuvent se trouver à la même hauteur. Dans ce cas, on ne trace la graduation correspondante qu'une seule fois.

****

DEXPLO

BUT:

Explore un arbre binaire dans l'ordre infixé et rend la valeur des cellules correspondant aux feuilles.

ACCES:

CALL DEXPLO(TREE,N,NOM)

PARAMETRES:

TREE [integer;array;input]

- tableau [1..n,1..3] contenant la représentation
de l'arbre binaire
précond: l'arbre est correctement construit

(afin d'éviter un bouclage)

N [integer;input]

- nombre de cellules de l'arbre
précond: n est impair

NOM [integer;output]

- tableau [1..(n+1)/2] des valeurs des feuilles

DISCUSSION:

La structure de l'arbre est discutée dans la procédure DCLASS.

****

DRAW

BUT:

Trace le dendrogramme correspondant à une classification hiérarchique.

ACCES:

```
CALL DRAW(NIDA,YPOS,XPOSG,XPOSD,YPOSG,YPOSD,PCCODE)
```

PARAMETRES:

NIDA [integer;input]

- nombre de groupes constituant le dendrogramme

YPOS [real;array;input]

- tableau [1..nida] des ordonnées des groupes
précond: ypos(i) sont classés dans l'ordre croissant

XPOSG,XPOSD [real;array;input]

- tableaux [1..nida] des abscisses gauche et droite, respectivement, des groupes constituant le dendrogramme

YPOSG,YPOSD [real;array;input]

- tableaux [1..nida] des ordonnées gauche et droite, respectivement, des groupes constituant le dendrogramme

PCCODE [integer;input]

- code graphique du dendrogramme

DISCUSSION:

Chaque groupe du dendrogramme est représenté par deux lignes verticales et une ligne horizontale formant ainsi

un U renversé.

Les variables xposg(i) et xposd(i) sont les abscisses des barres verticales. Les variables yposg(i) et yposd(i) sont les ordonnées des sous-groupes gauche et droit du groupe i.

****

FORMUL

BUT:

Calcule le maximum, le minimum ou la moyenne de deux nombres réels.

ACCES:

CALL FORMUL(NF,A,B,RES)

PARAMETRES:

NF [integer;input]

- numéro de la formule à appliquer
  précond: 1 < = nf < = 3

A,B [real;input]

- les deux nombres dont on cherche le maximum, le minimum ou la moyenne

RES [real;output]

- résultat de la comparaison de A et B

DISCUSSION:

Le choix est fait suivant la valeur de nf :

nf = 1 : on prend le maximum de a et b
nf = 2 : on prend le minimum de a et b
nf = 3 : on prend la moyenne de a et b

****

GADECO

BUT:

Décode le code graphique.

ACCES:

CALL GADECO(CODE,CL,ST,LT,BR)

PARAMETRES:

CODE [integer;input]

- code graphique

CL [integer;output]

- numéro de couleur
  postcond: 1 <= cl <= 8
  cl ≠ de la couleur de fond courante

ST [integer;output]

- numéro de style
  postcond: 1 <= st <= 4

LT [integer;output]

- indice de largeur de trait
  postcond: 0 <= lt <= 32767

BR [integer;output]

- indice de brillance du trait
  postcond: 0 <= br <= 32767

DISCUSSION:

La structure du code graphique a été donnée au point 2.4.1.1 .

PROGRAMMATION:

Les valeurs du 3 ème et du 4 ème chiffre sont utilisées pour calculer des indices variant entre 0 et 32767.

****

HCALC

BUT:

Calcule la hauteur des rectangles de l'histogramme, normalisée par la hauteur du plus grand rectangle.

ACCES:

CALL HCALC(REP,LIM,N,ORD)

PARAMETRES:

REP [integer;array;input]

- tableau [1..n] des répétitions par classe
  précond: rep(i) > 0

LIM [real;array;input]

- tableau [1..n + 1] des limites de classes
  précond: lim(i) strict. croissantes.

N [integer;input]

- nombre de classes

ORD [real;output]

- tableau des hauteurs normalisées
  postcond: 0 < ord(i) < = 1

****

HCONV

BUT:

Convertit les limites des classes de l'histogramme en abscisses réelles et en chaînes de caractères.

ACCES:

CALL HCONV(LIM,NLIM,XSIZE,ABSI,GRD)

PARAMETRES:

LIM [real;array;input]

- tableau [1..n + 1] des limites de classes
  précond: lim(i) strict. croissantes.

NLIM [integer;input]

- nombre de limites de classes

XSIZE [real;input]

- nombre d'unités de la fenêtre de vision en abscisses

ABSI [real;array;output]

- tableau [1..nlim] des abscisses des limites de classes
  postcond: 0 < absi(i) < xsize

GRD [integer;array of char.string;output]

- tableau [1..nlim] des graduations. Chaque élément de grd est une chaîne de 5 car. maximum

DISCUSSION:

La conversion est toujours effectuée par rapport à une fenêtre s'étendant de 0 à xsize en abscisses.

Les limites de classes sont converties en chaînes de caractères selon le format Fortran f5.2.
L'utilisateur doit tenir compte de ce fait dans la façon dont il fournit les limites.

****

HDECO

BUT:

Décode le code graphique de l'intérieur d'un polygone.

ACCES:

```
CALL HDECO(CODE,COLOR,SHADE)
```

PARAMETRES:

CODE [integer;input]

- code graphique de l'intérieur du polygone

COLOR [integer;output]

- couleur de l'intérieur du polygone
  postcond: 1 < = color < = 8
  color ≠ de la couleur de fond courante

SHADE [integer;output]

- style de l'intérieur du polygone
  postcond: 1 < = shade < = 6 ou shade = 47

DISCUSSION:

La variable shade désigne le type de hachures utilisé pour remplir le polygone.

La structure du code graphique de l'intérieur a été donnée en 2.4.1.1 .

****

HDRAW

BUT:

Trace l'ensemble des rectangles représentant les classes de l'histogramme.

ACCES:

```
CALL HDRAW(N,LIM,H,IND,ECODE,ICODE)
```

PARAMETRES:

N [integer;input]

- nombre de classes

LIM [real;array;input]

- tableau [1..n + 1] des limites de classes
  précond: lim(i) strict. croissantes.

H [real;array;input]

- tableau [1..n] des hauteurs normalisées des rectangles
  précond: 0 < h(i) <= 1

IND [integer;input]

- type de l'intérieur. Cette variable vaut 1 si l'intérieur des rectangles doit être hachuré et vaut 0 s'il doit être laissé vide

ECODE [integer;array;input]

- tableau [1..n] des codes graphiques du bord des rectangles. Le style des bords est toujours le trait plein.

ICODE [integer;array;input]

- tableau [1..n] des codes graphiques de l'intérieur des rectangles.

DISCUSSION:

Les valeurs du code graphique de l'intérieur des rectangles sont données au point 2.4.1.1 .

****

HPROJ

BUT:

Projette les valeurs des répétitions par classe de l'histogramme.

ACCES:

CALL HPROJ(REP,LIM,H,N,XSIZE,YSIZE)

PARAMETRES:

REP [integer;array;input]

- tableau [1..n] des répétitions par classe
  précond: rep(i) > 0

LIM [real;array;input]

- tableau [1..n + 1] des limites de classes
  précond: lim(i) strict. croissantes.

H [real;array;input]

- tableau [1..n] des hauteurs normalisées des rectangles
  précond: 0 < h(i) <= 1

N [integer;input]

- nombre de classes

XSIZE [real;input]

- nombre d'unités de la fenêtre de vision en abscisses

YSIZE [real;input]

- nombre d'unités de la fenêtre de vision en ordonnées

DISCUSSION:

La projection est toujours effectuée par rapport à une fenêtre s'étendant de 0 à xsize en abscisses et de 0 à ysize en ordonnées.

****

TITRE

BUT:

Trace une chaîne de caractères représentant le titre du dessin.

ACCES:

CALL TITRE(PCNAME,TCODE,DIR,XSIZE,YSIZE)

PARAMETRES:

PCNAME [integer;array of char.string;input]

- tableau [1..2] contenant le nom du dessin. Chaque élément de pcname est une chaîne de 5 car. maximum

TCODE [integer;input]

- code graphique du titre. Seule la couleur est prise en compte.

DIR [integer;input]

- direction de tracé du titre

XSIZE [real;input]

- nombre d'unités de la fenêtre de vision en abscisses

YSIZE [real;input]

- nombre d'unités de la fenêtre de vision en ordonnées

DISCUSSION:

Le titre est toujours tracé dans une fenêtre s'étendant de 0 à xsize en abscisses et de 0 à ysize en ordonnées.

Il apparaît sur le dessin en haut, au centre.

Le titre est tracé dans une des 4 directions suivantes:

de gauche à droite	(dir = 1)
de haut en bas	(dir = 2)
de droite à gauche	(dir = 3)
de bas en haut	(dir = 4)

****

TABLE OF CONTENTS